Titre: Title:	Algorithmes de diagnostic d'une chaîne JTAG reconfigurable et tolérante aux pannes au sein de la technologie WaferIC
Auteur: Author:	Safa Berrima
Date:	2014
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	Berrima, S. (2014). Algorithmes de diagnostic d'une chaîne JTAG reconfigurable et tolérante aux pannes au sein de la technologie WaferIC [Master's thesis, École Polytechnique de Montréal]. PolyPublie. https://publications.polymtl.ca/1636/

URL de PolyPublie: PolyPublie URL:	https://publications.polymtl.ca/1636/
Directeurs de recherche: Advisors:	Yvon Savaria, & Yves Blaquière
Programme: Program:	génie électrique

UNIVERSITÉ DE MONTRÉAL

ALGORITHMES DE DIAGNOSTIC D'UNE CHAÎNE JTAG RECONFIGURABLE ET TOLÉRANTE AUX PANNES AU SEIN DE LA TECHNOLOGIE WAFERIC

SAFA BERRIMA

DÉPARTEMENT DE GÉNIE ÉLECTRIQUE

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

MÉMOIRE PRÉSENTÉ EN VUE DE L'OBTENTION

DU DIPLÔME DE MAÎTRISE ÈS SCIENCES APPLIQUÉES

(GÉNIE ÉLECTRIQUE)

DÉCEMBRE 2014

UNIVERSITÉ DE MONTRÉAL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

Ce mémoire intitulé :

ALGORITHMES DE DIAGNOSTIC D'UNE CHAÎNE JTAG RECONFIGURABLE ET TOLÉRANTE AUX PANNES AU SEIN DE LA TECHNOLOGIE WAFERIC

présenté par : BERRIMA Safa

en vue de l'obtention du diplôme de : Maîtrise ès sciences appliquées

a été dûment accepté par le jury d'examen constitué de :

M. DAVID Jean-Pierre, Ph.D., président

M. SAVARIA Yvon, Ph.D., membre et directeur de recherche

M. BLAQUIÈRE Yves, Ph.D., membre et codirecteur de recherche

M. BOIS Guy, Ph.D., membre

REMERCIEMENTS

Je tiens à remercier le professeur Yvon Savaria, mon directeur de recherche et directeur du département de génie électrique de l'École Polytechnique de Montréal jusqu'à Juin 2014. Je le remercie pour ses judicieux conseils, l'aide et le temps qu'il a bien voulu m'accorder ainsi qu'avoir cru en moi pour me faire rejoindre son équipe de recherche. Je tiens à remercier également le professeur Yves Blaquière, mon co-directeur pour son écoute, son encadrement, sa patience et sa disponibilité malgré un emploi du temps chargé autant que directeur des programmes en microélectronique de l'université du Québec à Montréal.

J'ai eu beaucoup de chance d'avoir deux excellents encadrants tout au long de mon projet de maîtrise. Sans messieurs Yves Blaquière et Yvon Savaria ce travail n'aurait pas pu voir le jour.

Je remercie tous mes co-équipiers au laboratoire GRM de l'École Polytechnique de Montréal spécialement Gontran Sion et Sylvain Charasse. Grâce à leurs efforts, ce travail a pu être testé au laboratoire. J'ai bénéficié de leurs connaissances techniques et de leur savoir-faire.

Je veux exprimer mes remerciements et ma gratitude aux employés du département de génie électrique qui m'ont apporté l'aide et l'assistance nécessaires à l'élaboration de ce travail. Je remercie en particulier Réjean Lepage, analyste systèmes, pour son soutien informatique ainsi que Nathalie Lévesque, agente aux dossiers étudiants, qui a toujours été présente pour répondre efficacement à mes questions.

Je tiens finalement à exprimer ma gratitude envers ma famille pour tout l'amour et le soutien qu'ils m'apportent depuis toujours. J'espère qu'ils trouveront dans ce travail le fruit de leurs efforts et sacrifices.

RÉSUMÉ

Dans ce mémoire, des algorithmes de diagnostic d'une chaîne JTAG reconfigurable et tolérante aux pannes dans un circuit intégré à l'échelle de la tranche (*Wafer Scale Integrated Circuit WSI*) sont présentés. Le circuit intégré en question, nommé WaferIC, est au cœur du projet de recherche DreamWafer^{TM} qui implique plusieurs universités canadiennes. Ce projet vise à élaborer une plateforme de prototypage rapide pour les systèmes électroniques. C'est d'une certaine façon l'équivalent d'un circuit imprimé reprogrammable. Les circuits discrets, comme les FPGA et les mémoires par exemple seront déposés sur la surface du WaferIC. Ce dernier est un substrat programmable de la taille d'une tranche de Silicium et configurable qui réalise les interconnexions nécessaires entre les circuits et ce conformément à une spécification des interconnexions fournie par l'ingénieur en conception.

Le WaferIC est composé de milliers de cellules connectées entre elles par des liens intercellulaires formant ainsi un vaste réseau d'interconnexions reconfigurable. Une chaîne de balayage conforme au protocole JTAG est utilisée pour configurer les cellules du WaferIC.

Pour minimiser le temps de configuration, ce présent mémoire propose des algorithmes pour repérer le plus d'éléments (cellules et liens) fonctionnels possible au sein du WaferIC. La chaîne JTAG de configuration passera par ces éléments fonctionnels pour configurer toutes les cellules du WaferIC. Le premier objectif du diagnostic est d'établir un ensemble de chemins qui couvrent toutes les cellules et tous les liens intercellulaires du WaferIC. La taille des flux de bits JTAG qui créent ces chemins doit être minimale. Dans ce contexte, une étude théorique est faite dans ce mémoire pour prouver que la taille d'un flux de bits JTAG nécessaire pour établir un chemin de N cellules croit en $O(N^2)$.

Un algorithme de recherche basé sur le principe de la dichotomie a aussi été implémenté dans le cadre de ce projet de maîtrise. Cet algorithme est appliqué sur les chemins trouvés non fonctionnels pour localiser le plus précisément possible les liens défectueux dans ces chemins. L'état des cellules sera déduit à partir des liens. En effet, une cellule est défectueuse si tous ses liens entrants ou sortants sont défectueux. Des algorithmes heuristiques ont également été implémentés pour analyser les chemins non fonctionnels dans le cas où l'algorithme de

dichotomie n’arrive pas à localiser précisément le(s) lien(s) défectueux. Ces algorithmes heuristiques améliorent la résolution du diagnostic et tentent de localiser le plus étroitement possible la ou les défectuosité(s).

Les algorithmes développés ont été testés sur un prototype miniaturisé du WaferIC. Dans un réticule contenant 1024 cellules, une zone de 4 cellules a été trouvée potentiellement défectueuse, le reste du réticule a été caractérisé comme fonctionnel par les algorithmes de diagnostic. Des résultats de simulation ont également été dressés montrant le comportement des algorithmes en présence d’un ou de plusieurs liens défectueux. L’algorithme de dichotomie développé localise un seul lien défectueux dans le réticule avec une probabilité de plus de 99%. Il localise jusqu’à huit liens défectueux simultanés avec une probabilité de 71%.

ABSTRACT

In this master project, algorithms to diagnose a reconfigurable and defect tolerant JTAG scan chain in a wafer scale integrated circuit are proposed. The integrated circuit, called WaferIC is at the core of the DreamWafer^{TM} research project involving several Canadian universities. This project aims to develop a platform for rapid electronic system prototyping. That platform is analogous to a reconfigurable printed circuit board. Circuits are deposited on the surface of the WaferIC. This device is a configurable and programmable substrate that implements all the necessary interconnections between the circuits in accordance with the user specification.

The WaferIC is made of thousands of cells interconnected with intercellular links forming an extensive and reconfigurable network of interconnections.

A JTAG scan chain is used to configure the cells of the WaferIC. To minimize the configuration time, this master project proposes algorithms to identify functional elements (cells and links). This scan chain uses those functional elements to configure all the cells of the WaferIC. The first objective is to find a set of paths that cover all cells and links of the WaferIC. The length of the JTAG bit streams that create these paths must be reasonably short, and possibly optimal. In light of this, a theoretical study is done that proves that the size of a JTAG bit stream grows as $O(N^2)$ for a path made of N cells. A set search dichotomic algorithm was also developed to be applied on defective paths to accurately locate defective links within these paths. The state of cells can be deduced from links. Indeed, if all incoming and outgoing links of a cell are defective, then the cell is defective. Heuristic algorithms have also been developed to analyze non-functional paths in the case where the dichotomic algorithm is unable to locate precisely the defective link(s).

The algorithms developed were tested on a miniaturized prototype of the WaferIC. In a reticle containing 1024 cells, an area of 4 cells has been found as potentially defective. The rest of the reticle has been characterized as functional by the diagnosis algorithms. Simulation results were also been reported showing the behavior of the algorithm in presence of one or several defects. It is shown that the dichotomic algorithm accurately pinpoints a single defect with a

probability of more than 99% and also accurately locates up to 8 defects with a probability of 71%.

TABLE DES MATIÈRES

LISTE DES TABLEAUX

LISTE DES FIGURES

LISTE DES SIGLES ET ABRÉVIATIONS

BGA	Ball Grid Array
CDSP	Circuit Discret Sous Prototypage
FPGA	Field Programmable Gate Array
IC	Integrated Circuit
JTAG	Joint Test Action Group
LAIC	Large Area Integrated Circuit
MCM	Multi-Chip Module
PCB	Printed Circuit Board
SiP	System in Package
SoC	System on Chip
SoW	System on Wafer
TSV	Through Silicon Via
UUT	Unit Under Test
VLSI	Very Large System Integration
WSI	Wafer Scale Integration

LISTE DES ANNEXES

INTRODUCTION

Les systèmes électroniques actuels, à la fine pointe de la technologie, sont des systèmes très complexes contenant parfois un grand nombre de puces électroniques. Les puces sont traditionnellement montées et interconnectées moyennant une carte imprimée appelée PCB (*Printed Circuit Board*). Rassembler plusieurs puces sur une carte imprimée est une tâche laborieuse. En effet, plusieurs difficultés surgissent. On peut citer par exemple :

- Plus le nombre de puces est élevé, plus la taille du circuit global est grande. Ceci augmente le coût de fabrication.
- Pour interconnecter un grand nombre de modules sur une petite surface, il faut miniaturiser la taille des composants électroniques. Ceci est limité par des contraintes physiques.
- Dans les PCB très dense, la phase de test est une tâche ardue vu que certaines broches sont inaccessibles (typiquement sous les puces quand on utilise des boîtiers de haute densité comme les BGA (*Ball Grid Array*)).
- Afin de répondre à la demande d'intégration des systèmes électroniques dans des volumes de plus en plus petits, les PCB actuels comportent plusieurs couches de métallisation. Ceci rend non seulement les processus de routage et de test plus difficiles, mais aussi amène des coûts plus élevés lors de la fabrication.
- Plus le système électronique réalisé est dense, plus il est difficile de résoudre les problèmes qui peuvent surgir, que ce soit au niveau matériel ou logiciel. Le test et la validation de tels systèmes deviennent donc assez longs et coûteux.

Devant les difficultés rencontrées avec les PCB, le prototypage rapide à l'aide d'un circuit intégré à l'échelle de la tranche (*Wafer Scale Integrated* (*WSI)*) est donc apparu comme une alternative intéressante à considérer. C'est dans le cadre des circuits intégrés à l'échelle de la tranche que Richard Norman a proposé une solution [4] à l'ensemble des problèmes cités. Il introduit un nouveau concept de circuit intégré nommé le « WaferIC ». C'est un circuit intégré à l'échelle de la tranche de Silicium capable d'interconnecter un très grand nombre de plots sur un ensemble de circuits électroniques ayant chacun jusqu'à 2000 plots et même plus. C'est en quelque sorte un substrat intelligent permettant de connecter des puces déposées sur sa surface suivant une

spécification des interconnexions communément appelée *netlist* fournie par l'utilisateur. Ces interconnexions sont reprogrammables autant de fois que l'utilisateur le souhaite. Cette solution brevetée par Norman est la propriété de Gestion TechnoCap Inc et cette technologie est nommée « WaferBoard^{TM}».

Le WaferBoard^{TM} est une plateforme de prototypage rapide des systèmes numériques. Cette plateforme a été prototypée dans le cadre d'un projet de recherche nommé DreamWafer ^{TM} réalisé en coopération entre plusieurs universités canadiennes (notamment Polytechnique Montréal, Université du Québec à Montréal, Université du Québec en Outaouais et McGill) ainsi qu'en compagnie de Gestion TechnoCap Inc société basée à Montréal. Plusieurs étudiants et professeurs ont contribué au projet depuis 2006 chacun dans son domaine.

La plateforme WaferBoard^{TM} offre un réseau d'interconnexion reprogrammable. Les interconnexions entre les puces sont effectuées à l'aide de la plateforme conformément à une spécification des interconnexions désirées communément appelée une *netlist* fournie par l'utilisateur. Le WaferBoard^{TM}, une fois mis sur le marché, permettrait de réduire le coût de prototypage ainsi que le temps de commercialisation des systèmes électroniques. Le WaferBoard^{TM} permettrait aux ingénieurs logiciels de travailler sur un système fonctionnel pendant que les ingénieurs matériels en conçoivent le circuit imprimé. Ceci permettrait de réaliser un gain économique non négligeable.

Les puces à interconnecter, qu'on appelle aussi CDSP (Circuit Discret Sous Prototypage), sont déposées sur la surface active du WaferIC. Le dépôt des puces est insensible à l'alignement, c'est-à-dire que les puces sont déposées dans n'importe quel sens. Le WaferIC, breveté par Monsieur Richard Norman, comporte 76 images de réticule, où chaque réticule est une mer de cellules identiques au niveau physique. Ces cellules sont interconnectées par des liens intercellulaires formant ainsi un vaste réseau d'interconnexions. Chaque cellule contient des petits plots appelés « *NanoPads* » qui peuvent alimenter en tension les broches des puces avec lesquelles ils sont en contact.

Pour obtenir les interconnexions nécessaires entre les puces déposées, il faut programmer les cellules du WaferIC qui sont en contact avec ces puces. La programmation du WaferIC se fait moyennant une chaîne conforme à un sur-ensemble de la norme de test JTAG [1]. Une seule

chaîne commence par le port d'entrée du contrôleur de test, configure toutes les cellules (qui sont en contact avec les puces) puis ressort par le port de sortie du contrôleur de test. La chaîne JTAG est donc composée d'un ensemble de cellules et de liens intercellulaires. La chaîne est un sur-ensemble de JTAG qui est reconfigurable (c'est-à-dire qu'elle peut être reprogrammée pour parcourir n'importe quelle cellule) et tolérante aux pannes. Ces deux caractéristiques ne sont pas présentes dans des chaînes JTAG standard. Deux aspects de la tolérance aux pannes qui supportent la chaîne JTAG étudiée dans ce mémoire sont à mentionner :

- Si une cellule particulière est défectueuse, elle peut toujours être programmée (configurée) à partir d'une cellule voisine fonctionnelle.
- La chaîne JTAG ne peut passer que par des éléments (cellules et liens) dont la fonctionnalité est validée. En effet, si la chaîne JTAG rencontre ne serait-ce qu'une seule défectuosité, elle sera brisée et complétement inutilisable.

Ces deux aspects de la tolérance aux pannes nécessitent un diagnostic préalable pour repérer les éléments (cellules et liens) fonctionnels du WaferIC.

Objectif :

Le diagnostic de la chaîne JTAG est le sujet de recherche de cette maîtrise. Ce mémoire présente des algorithmes et des heuristiques qui ont été développés avec le langage C++ pour diagnostiquer un réticule du WaferIC. Ces algorithmes peuvent être appliqués sur tous les réticules diagnostiquant ainsi la totalité de la surface active du WaferIC. Le but du diagnostic est de trouver le plus d'éléments fonctionnels possible pour pouvoir configurer la totalité des cellules du réticule.

Méthodologie :

L'idée de base est d'injecter des flux de bits conformes à la norme JTAG étendue selon nos besoins au sein du réticule afin d'établir un ensemble de « chemins » qui seront utilisés par l'algorithme de diagnostic. Les chemins à établir doivent satisfaire à deux conditions :

- Le flux de bits JTAG nécessaire à la création du chemin devrait demeurer petit et de taille acceptable. Pour cela une analyse de complexité temporelle a été dressée dans ce mémoire et

elle en représente une contribution théorique. Cette analyse a montré que la taille d'un flux de bits JTAG nécessaire pour créer (établir) un chemin de N cellules est de $2N^2+18N$ bits.

- L'ensemble des chemins doit couvrir la totalité du réticule (c'est-à-dire que chaque cellule et chaque lien est parcouru par au moins un chemin).

Ces deux conditions nous ont amené à comparer deux manières possibles pour couvrir l'ensemble des cellules du réticule : La méthode basée sur des chemins rectangulaires et la méthode basée sur des chemins en serpentins. Suite à l'analyse de complexité temporelle élaborée, la couverture basée sur des chemins rectangulaires a été choisie.

Le diagnostic se fait en trois étapes successives :

Étape 1: Des données sont envoyées à travers les chemins crées (rectangulaires) et la sortie de chacun est comparée à ce qui est attendu. Si les données en sortie correspondent à ce qui est attendu, alors tous les éléments (cellules et liens) constituant le chemin sont caractérisés, par l'algorithme, comme fonctionnels, sinon il existe un ou plusieurs éléments défectueux dans le chemin. Ces chemins non fonctionnels sont sauvegardés dans une liste.

Étape 2 : Afin d'élargir l'espace des éléments fonctionnels, un algorithme de recherche basé sur le principe de la dichotomie est déployé pour localiser le(s) élément(s) défectueux dans les chemins trouvés non fonctionnels (les chemins sauvegardés dans la liste). La recherche de l'élément défectueux se fait au niveau des liens intercellulaires. En effet, l'état des cellules sera déduit à partir des liens : une cellule défectueuse est une cellule ayant tous les liens entrants ou sortants défectueux.

Étape 3 : Des algorithmes heuristiques ont également été développés dans le cas où le lien défectueux n'est pas étroitement repéré par l'algorithme de dichotomie. Ces algorithmes ont pour but d'améliorer la résolution du diagnostic.

Les algorithmes de diagnostic développés ont été testés sur un prototype miniaturisé du WaferIC. Parmi les 1024 cellules du réticule, seulement une zone comportant 4 cellules a été identifiée comme étant une zone potentiellement défectueuse. Le reste du réticule a été trouvé fonctionnel. Des résultats de simulation sont également dressés montrant le comportement de l'algorithme en présence d'un et de plusieurs liens intercellulaires défectueux.

Ce mémoire comporte quatre chapitres et une conclusion :

- Le chapitre 1 est une revue de littérature. Il décrit en détail les fonctionnalités de la plateforme WaferBoard^{TM} ainsi que l'architecture du WaferIC. Dans ce chapitre on y trouve également l'architecture de la chaîne JTAG reconfigurable du WaferIC.
- Le chapitre 2 présente l'analyse de complexité temporelle pour configurer (créer) un chemin de *N* cellules ainsi qu'une comparaison entre deux méthodes pour obtenir une couverture du réticule : La méthode par des chemins en serpentins et la méthode par des chemins rectangulaires.

  Dans ce chapitre une modélisation des chemins rectangulaires avec le logiciel MATLAB a également été faite.
- Le chapitre 3 présente en détail l'algorithme de recherche de(s) lien(s) défectueux basé sur le principe de la dichotomie. On y trouve également les algorithmes heuristiques développés dans le cas où l'algorithme de dichotomie n'arrive pas à localiser étroitement le lien défectueux.
- Le chapitre 4 présente une étude basée sur des lois probabilistes pour estimer au mieux le nombre de défectuosités qui peuvent être présentes dans un réticule du WaferIC. Ce chapitre résume également les résultats de simulation ainsi que les résultats expérimentaux des algorithmes développés.
- Finalement une conclusion résumant les principales contributions de cette recherche est dressée.

CHAPITRE 1 REVUE DE LITTÉRATURE

1.1 Introduction

Ce présent chapitre est une revue de littérature. Il est constitué essentiellement de cinq sections. La première section traite de quelques limitations des systèmes basés sur des cartes imprimées (PCB) ainsi que les solutions qui sont apparues pour résoudre au mieux ces limitations. Parmi ces solutions on cite les MCM, les SOC, les SIP et les circuits intégrés à l'échelle de la tranche. C'est dans le contexte des circuits intégrés à l'échelle de la tranche qu'est apparu le concept de la technologie WaferIC qui est au cœur du projet de recherche DreamWafer^{TM}. Ce projet vise à élaborer une plateforme de prototypage rapide des systèmes électroniques nommée WaferBoard^{TM}. Cette plateforme est décrite en détail dans la section 2. Étant donné que la base de ce projet de maîtrise est le diagnostic de la chaîne JTAG au sein du WaferIC, une présentation du protocole JTAG est introduite dans la troisième section. On y trouve une brève description de cette norme ainsi que quelques architectures des chaînes de balayage existantes et quelques techniques de tolérance aux pannes. La quatrième section traite de l'architecture de la chaîne JTAG adoptée au sein du projet afin de configurer et tester les cellules du WaferIC. Finalement, dans les cinquième et sixième sections, la problématique est posée et les contributions de ce mémoire sont soulignées. Les grands titres de la contribution sont l'élaboration d'une analyse de complexité temporelle de diagnostic et le développement des algorithmes de diagnostic de la chaîne JTAG au sein du WaferIC.

1.2 Les limitations des systèmes sur PCB

Depuis la découverte du transistor par William Shockley et ses collègues en 1947, la microélectronique n'a pas cessé d'évoluer. Des systèmes de plus en plus complexes et performants sont conçus. Traditionnellement, pour réaliser un système électronique assez complexe, plusieurs circuits intégrés sont rassemblés et interconnectés sur une carte imprimée appelée PCB (*Printed Circuit Board*). Néanmoins plus le système à concevoir est complexe, plus l'utilisation des PCB est laborieuse. En effet, les PCB présentent un certain nombre d'inconvénients, on peut citer par exemple que :

- Plus le nombre de puces à interconnecter est élevé, plus la taille du circuit global est grande et plus le coût de fabrication est élevé.

- Dans les PCB très denses, la phase de test est généralement une tâche ardue. En effet, il est parfois impossible de tester certaines parties du design parce qu'elles sont inaccessibles et cachées sous des couches de métallisation.
- Le routage est difficile à faire dans les PCB très denses.
- Les PCB consomment de l'énergie à cause des délais dans les connexions inter-puces.
- Le temps de conception ainsi que les délais de fabrication d'un système sur PCB est assez long. Les coûts associés aux délais et aux itérations quand un PCB doit être refait plusieurs fois sont considérables.
- Les opérations de déverminage et de test sont difficiles dans un PCB très dense.

Pour pallier les problèmes rencontrés avec les PCB, quelques solutions sont apparues. On peut citer par exemple :

- Les MCM (*multi-chip modules*) où plusieurs puces et circuits intégrés sont montés sur un même substrat. L'ensemble est considéré comme étant un seul module. Ce module résultant sera ensuite soudé sur un PCB afin d'améliorer la densité et la performance. Plusieurs technologies modernes utilisent les MCM comme les processeurs POWER5 et POWER7 d'IBM.
- Les systèmes sur puce (*Systems on chip* SOC) [2]. Cette approche permet de rassembler un ensemble de fonctionnalités diverses sur une même puce. Les composants peuvent être des mémoires, des composants passifs (résistance, capacité), des microprocesseurs, des circuits logiques, analogiques etc… Bien que les systèmes sur puces augmentent considérablement la densité et permettent des gains en performances, connecter des composants qui ne sont pas de même nature peut être assez complexe et coûteux.
- L'approche SIP (*System in package*) permet de rassembler plusieurs puces dans un même boîtier. L'intégration 3D est une forme plus récente des SIP. Elle permet d'occuper moins d'espace et d'augmenter la vitesse et la performance des systèmes électroniques en empilant plusieurs puces verticalement (les unes sur les autres) et en assurant leurs interconnexions à travers des TSV (*Through Silicon Vias*). Cette solution reste limitée par le nombre de puces à empiler.

Les circuits intégrés à l'échelle de la tranche ont vu le jour dans les années 1970 lorsque des technologies à base de semi-conducteurs ont commencé à se développer. Une des promesses des circuits à l'échelle de la tranche est une amélioration de la fiabilité des systèmes complexes à cause d'une réduction du nombre de connexions externes généralement moins fiables que les interconnexions intégrées. En 2009, le système sur tranche (*System on Wafer* SoW) [3] a été introduit. Avec ce concept, les circuits intégrés sont interconnectés à l'aide d'un substrat qui peut être aussi grand qu'une tranche complète qui conduit à la technique dite WLCSP (*Wafer Level Chip Scale Package*). Les avantages majeurs par rapport aux PCB conventionnels, sont l'élimination de plusieurs connexions inter-puces, une augmentation de la densité et une diminution des délais et de la dissipation en lien avec les interconnexions. Dans ce cas, un substrat de Silicium sert de support mécanique et effectue également les interconnexions nécessaires entre les divers composants.

C'est dans le contexte des circuits intégrés à l'échelle de la tranche que le brevet de Richard Norman [4] a été développé en 2006. Ce brevet présente plusieurs innovations en lien avec les circuits intégrés à grande surface, aussi appelés *Large Area Integrated Circuits* (LAIC). Le LAIC qui fait l'objet de recherches dans la présente maîtrise est nommé WaferIC. Les circuits à l'échelle de la tranche rassemblent plusieurs dizaines ou centaines de millions de transistors. Il existe plusieurs types de LAIC. On peut citer par exemple le LAIC monolithique nommé MAXPE9 [5]. Il contient neuf processeurs et sa taille est de 16.6 cm^2. Ce circuit est dédié pour des applications de codage vidéo en temps réel. Plus tard, un autre LAIC monolithique [6] a été proposé. Il contient 16 processeurs et 64 Mbits de DRAM. En 2011, un circuit intégré à l'échelle de la tranche à base de transistors en graphène [7] a été proposé. Grâce aux propriétés thermiques et électriques du graphène, de tels circuits intégrés permettraient possiblement d'augmenter la fréquence d'opération des systèmes jusqu'à 400GHz. Il existe également des LAIC optoélectroniques constitués de pixels ou bien des LAIC constitués d'éléments capteurs. On trouve aussi des LAIC constitués de réseaux de neurones analogiques [8]. Contrairement aux LAIC précédents [5-8], le WaferIC n'a pas pour but d'améliorer la puissance de calcul ou bien la fréquence d'opération, mais plutôt de permettre une densité d'interconnexion très importante. En effet, c'est un substrat intelligent permettant d'interconnecter plusieurs circuits intégrés, chaque circuit pouvant avoir jusqu'à 2000 broches. Le WaferIC est né dans le cadre du projet de

recherche DreamWafer^{TM} qui vise à élaborer une plateforme de prototypage rapide des systèmes électroniques. La plateforme est nommée WaferBoard ^{TM}.

1.3 Le WaferBoard^{TM}

La figure 1-1 illustre les quatre principaux blocs de la plateforme WaferBoard ^{TM} : Le WaferIC, le PCB supérieur (*Top PCB*), le module de puissance (*Power supply*) et le PCB inférieur (*Bottom PCB*). Le PCB supérieur fournit une interface externe pour communiquer avec un ordinateur tandis que le PCB inférieur distribue la puissance. La plateforme est constituée d'une tranche entière de Silicium (WaferIC) de 200 mm de diamètre. Avec cette technologie sous-développement, l'utilisateur dépose simplement des circuits intégrés (IC) sur la surface active (WaferIC) puis la plateforme détecte automatiquement les positions des broches et les envoie à l'utilisateur. Ensuite moyennant un logiciel dédié à ce projet, nommé WaferConnect l'utilisateur retourne au WaferIC une *netlist* contenant les interconnexions à faire, que ce soit entre les circuits intégrés eux-mêmes ou entre les circuits et le substrat. Ainsi le réseau d'interconnexion au sein du WaferIC sera configuré selon la demande de l'utilisateur.

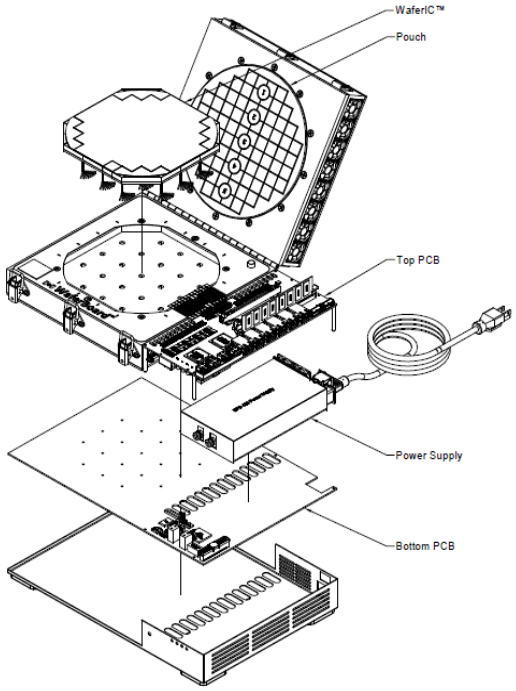


Figure 1-1: Modèle éclaté du WaferBoard ^{TM} (figure extraite de [9]).

Que les composants électroniques soient de la nouvelle technologie ou non, le WaferBoard™ simplifie le prototypage de tels systèmes électroniques en fournissant tout ce dont les divers composants ont besoin : alimentation, masse, connexions entre les puces et avec le monde externe. Un tel système peut être rapidement testé et modifié. Le WaferBoard™, tel qu'envisagé, permettrait ainsi de réduire considérablement les coûts de développement et de déverminage des systèmes électroniques. Ultimement, cette technologie promet de réduire les temps requis dans les itérations de conception et validation des systèmes complexes.

Le mode d'utilisation du Waferboard™ est assez simple. Le Waferboard™ ressemble à un boîtier. Une fois que le couvercle du boîtier est ouvert, les composants électroniques peuvent être déposés sur le WaferIC. Le système est insensible à l'alignement, c'est-à-dire que les composants peuvent être déposés de façon approximative et pas parfaitement alignés avec les connexions internes. Une fois les composants déposés, le couvercle est bien fermé pour exercer de la pression sur les circuits afin de les maintenir en place. Le WaferIC est un substrat actif, qui comporte des circuits permettant de détecter la présence des billes des puces électroniques et de repérer leurs emplacements. Une fois les billes détectées, la plateforme peut reconnaître les composants déposés sur sa surface si ceux-ci sont dans un catalogue de circuits connus, ce qui est le cas pour la majorité des circuits intégrés présents sur le marché. La plateforme envoie à l'utilisateur, via le logiciel de configuration WaferConnect, une carte permettant de visualiser l'emplacement des circuits et des broches ainsi que des propositions sur la manière de placer les circuits déposés. L'utilisateur confirme l'identité des circuits déposés et spécifie les connexions entre les circuits, que ce soit par la souris ou en important une *netlist*. Le logiciel WaferConnect permet de programmer la plateforme pour implémenter les interconnexions telles que demandées par l'utilisateur. Le prototype est ainsi prêt à fonctionner.

Dans les paragraphes suivants, les principaux éléments du WaferBoard™, à savoir le WaferIC, le WaferNet, le WaferConnect et les PowerBlocks sont décrits en détail.

1.3.1 WaferIC

La figure 1-2 décrit l'architecture interne du WaferIC. C'est un circuit intégré à l'échelle de la tranche crée par la photo-répétition de soixante-seize (76) images de réticules par le principe de la photo-lithographie [10]. Chaque réticule contient 1024 (32×32) cellules toutes identiques au niveau physique et ne sont différenciées que par le contenu de leurs mémoires internes. Au total,

le WaferIC contient 77 824 cellules. Chaque cellule contient 16 (4×4) plots conducteurs très fins appelés *NanoPads* qui fournissent aux circuits intégrés une tension Vdd stable et régulée ou bien la masse dont ils ont besoin. Le WaferIC contient au total 1, 245,184 *NanoPads*. Grâce à ces *NanoPads*, le WaferIC permet d'alimenter en tension les puces déposées sur sa surface.

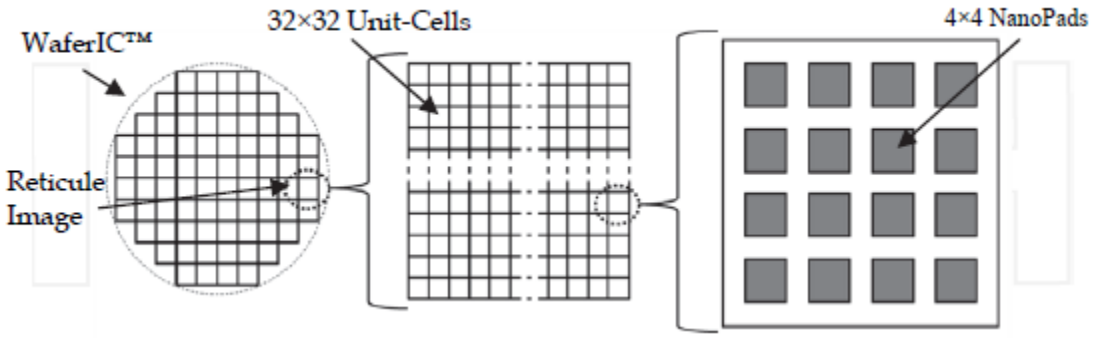


Figure 1-2: Zoom au niveau d'un réticule et d'une cellule (figure extraite de [9]).

La base de ce substrat programmable repose sur une technique nommée le "*reticle stiching*"[11]. Cette technique consiste à interconnecter les réticules voisins par une couche de métal définie à partir des masques de photolithographie pour assurer les connexions entre les différents réticules. La surface du WaferIC est égale à 24 500 mm^2 et elle est couverte par un tissu conducteur anisotropique appelé film axe- Z (en anglais *Z-axis film*) [12]. Ce film adhésif contient près de 80 millions de fibres conductrices (64 fibres par *NanoPad*) afin d'établir un contact électrique entre les *NanoPads* et les billes des puces déposées sur la surface du WaferIC.

1.3.2 WaferNet

Le WaferNet est le réseau interne reconfigurable du WaferIC. Il a été conçu pour supporter les circuits intégrés les plus utilisés, à savoir les processeurs, les FPGA et les mémoires. C'est en quelque sorte un réseau maillé multidimensionnel qui peut faire un très grand nombre d'interconnexions. La figure 1-3 illustre les interconnexions entre les cellules au sein du WaferNet.

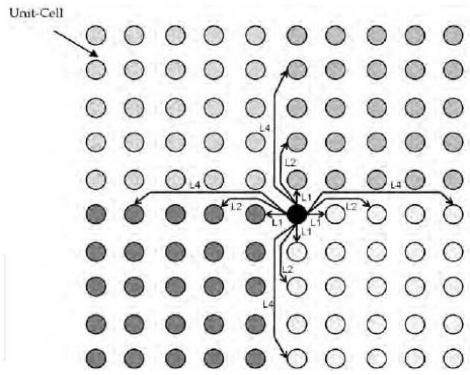


Figure 1-3: Interconnexions au sein du WaferNet (figure extraite de [9]).

Une cellule est liée à sa voisine immédiate mais aussi à des cellules distantes de 2, 4, 6, 8… cellules et ceci dans les quatre directions (Nord, Est, Ouest, et Sud). Chaque cellule contient dans son architecture interne des répéteurs pour préserver l'intégrité du signal des nombreuses interconnexions qui la traverse. Ces répéteurs servent aussi à minimiser les temps de propagation sur les longues interconnexions. Les cellules sont reliées entre elles par ce qu'on appelle des liens intercellulaires. Chaque cellule est dotée d'une porte logique OU en entrée pour sélectionner un lien entrant et d'un démultiplexeur en sortie pour rediriger un lien sortant vers une cellule de destination. Puisque le WaferIC est configuré en utilisant la norme JTAG [1], chaque cellule contient en dehors de sa logique interne, un contrôleur TAP. L'architecture interne d'une cellule unitaire est décrite plus loin dans ce chapitre.

1.3.3 PowerBlock

La plateforme contient 21 modules d'alimentation appelés Power blocks qui sont situés sous le WaferIC pour assurer la régulation de la tension. Un Power Block est un PCB de taille 35mm × 35mm qui alimente en tension le WaferIC par l'intermédiaire d'un régulateur de tension (Figure 1-4). Il assure également la communication entre le WaferConnect (logiciel dédié au projet) et le WaferIC moyennant le protocole JTAG et par l'intermédiaire d'un FPGA.

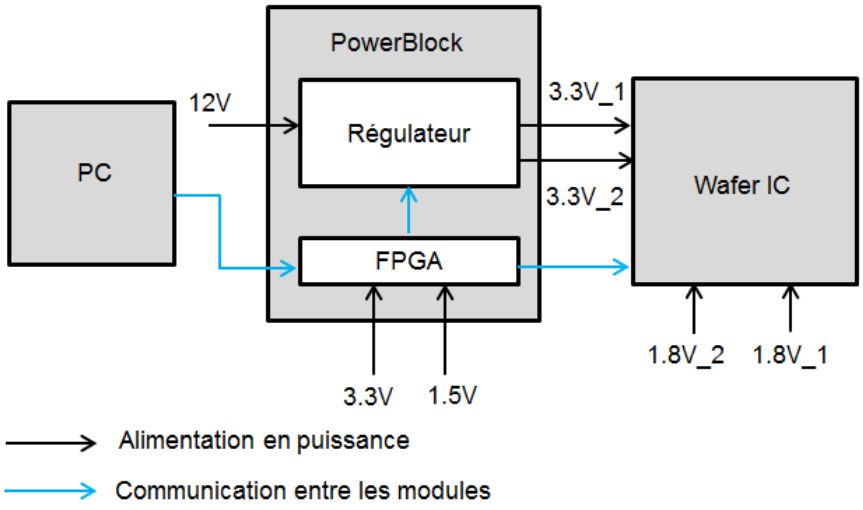


Figure 1-4: Alimentations requises au PowerBlock et au WaferIC.

On retrouve ainsi, sur chaque réticule, cinq « ports » correspondants aux signaux JTAG :

• TCK : signal d'horloge

• TRST : reset

• TMS : signal de commande

• TDI : données entrantes

• TDO : données sortantes

L'alimentation en puissance (VDD3.3, VDD1.8 et GND) et les cinq fils JTAG parviennent au réticule par des trous gravés dans le silicium appelés TSV (*Through Silicon Vias*). Chaque réticule contient 64 TSV. Au total, le WaferIC nécessite 4864 TSV.

1.3.4 WaferConnect

Afin d'exploiter les fonctionnalités de la plateforme, une suite d'outils logiciels appelée WaferConnect [13] a été mise en place.

Le flux de travail du logiciel WaferConnect se compose de sept étapes :

1. Démarrer et diagnostiquer la plateforme.
2. Détecter les points de contact Broche-*NanoPad*.
3. Reconnaitre les circuits déposés.
4. Envoyer une *netlist*.
5. Calculer la longueur des interconnexions.

6. Configurer le réseau.
7. Générer les rapports.

La présentation qui suit élabore brièvement sur chacune de ces étapes.

Démarrer et diagnostiquer la plateforme

Une fois que les circuits à interconnecter sont déposés sur le WaferIC, la plateforme est alors alimentée en tension et la phase de diagnostic commence automatiquement. La phase de diagnostic a pour but de localiser les défectuosités qui peuvent surgir lors de la dernière utilisation de la plateforme.

Détecter les points de contact broche-*NanoPad*

Une fois les circuits intégrés déposés sur le WaferIC et que la plateforme a été mise sous tension, le logiciel WaferConnect détecte les contacts entre les broches des circuits à interconnecter et les *NanoPads*. En effet, puisque le WaferIC contient 1 245 184 *NanoPads* de très petite taille une broche déposée sur le WaferIC est en contact avec plusieurs *NanoPads*. Donc une broche crée un court-circuit entre les *NanoPads* sur lesquels elle se pose. Pour détecter les contacts broche-*NanoPad*, chaque *NanoPad* contient un capteur pour repérer les courts circuits avec les *NanoPads* voisins. À peu près un million de détecteurs de court-circuit sont implémentés dans le WaferIC.

Reconnaitre les circuits déposés

Un outil de reconnaissance de boîtiers peut à l'aide des informations recueillies lors de la détection des broches et en comparant ces informations avec celles des circuits présents dans une bibliothèque, identifier de façon certaine, ou faire des suggestions à l'utilisateur que ce dernier devrait valider pour confirmer la nature de chacune des puces détectées. Dans le cas où le boîtier (*package*) n'est pas reconnu par le WaferConnect, il est possible d'assigner les broches et d'enregistrer l'information pour une utilisation future.

Envoyer une *netlist*

L'utilisateur fournit à la plateforme, à travers le logiciel WaferConnect, une *netlist* contenant les interconnexions à faire entre les différents circuits intégrés déposés sur la surface WaferIC ainsi

que les contraintes à respecter. La *netlist* peut être définie manuellement ou bien importée (par exemple d'un fichier GRB, EDIF, etc.).

Calculer les longueurs des interconnexions

Après que la *netlist* soit envoyée à la plateforme, WaferConnect peut calculer la longueur des routes pour configurer les cellules et il peut éventuellement proposer dans certains cas de déplacer certains circuits afin de réduire la longueur des interconnexions.

Configurer le réseau

Une chaîne JTAG est envoyée vers la plateforme en passant par un FPGA afin de configurer les cellules qui sont en contact avec des broches. La chaîne JTAG commence par la cellule connectée physiquement au port TDI, parcourt les cellules à configurer, puis se termine par la cellule connectée au port TDO. Lors de la configuration, la chaîne doit éviter de passer par les éléments défectueux repérés lors de l'étape de diagnostic.

Générer les rapports

À la fin, des rapports sont générés pour permettre à l'utilisateur de vérifier si le prototypage a été fait conformément aux spécifications demandées ou pas.

1.4 Protocole de test avec les chaînes JTAG

1.4.1 Brève introduction à la norme JTAG

Étant donné que le test, le diagnostic ainsi que la configuration du WaferIC se basent sur le protocole JTAG, un survol rapide de ce protocole est nécessaire. Dans ce paragraphe, un bref résumé du JTAG est dressé. Pour s'approfondir dans le sujet, le lecteur peut toujours se référer à la dernière version de la norme [1] sortie en 2013 et qui contient de légères modifications par rapport à l'ancienne version [14].

Le standard JTAG (*Joint Test Action Group*) appelé aussi « *Boundary scan* » a été normalisé en 1990 dans le but de faciliter le test des PCB. Aujourd'hui cette norme est utilisée pour tester et programmer des systèmes électroniques de plus en plus complexes. Chaque broche du circuit intégré est connectée à une « cellule JTAG » pour être testée. Ces cellules sont interconnectées par un seul bus formant un registre à décalage. Des données sont envoyées depuis le contrôleur de test à travers le port TDI, parcourent les cellules JTAG et ressortent par le port TDO. Si la

séquence de données au niveau de TDO correspond à la séquence attendue, alors tous les circuits sous test sont fonctionnels. Un tel registre à décalage peut être aussi utilisé pour programmer des FPGA ou des microcontrôleurs. Cinq signaux de contrôle sont à la base du protocole JTAG :

- TMS (Test Mode Select) : C'est le signal qui commande les cellules JTAG. Il sélectionne le mode sous lequel la cellule fonctionne.
- TRST (Test Reset) : C'est un signal de réinitialisation (optionnel).
- TDI : Les données en entrée.
- TDO : Les données en sortie.
- TCK : L'horloge du système.

Plusieurs circuits intégrés compatibles avec la norme JTAG peuvent être interconnectés et testés en liant le TDO sortant d'un circuit au TDI entrant du circuit voisin. Une telle liaison crée une « chaîne JTAG ».

1.4.2 Différentes architectures de chaînes JTAG existantes

La base de ce projet de maîtrise est le diagnostic des défectuosités au sein d'une chaîne JTAG. Diagnostiquer revient à localiser les pannes détectées lors de la phase de test. Les techniques de test sont donc à la base du diagnostic. Puisque ce projet repose sur une architecture spéciale de la chaîne de balayage, il est donc pertinent de faire un survol rapide des différentes architectures des chaînes de balayage utilisées pour des fins de test.

Plusieurs types de chaînes de balayage existent comme la chaîne sérielle complète (en anglais *Full scan chain*) où tous les registres (*Flip-flops*) du circuit sont inclus dans la chaîne. Un problème avec cette chaîne élémentaire, c'est qu'elle peut nécessiter un temps de test relativement long, puisque les données doivent être décalées dans tous les registres du circuit. Comme alternative à l'architecture de chaîne complète, on trouve l'architecture « *Shadow* » où les registres sont dupliqués. Un ensemble de registres est utilisé pour le mode normal du circuit tandis que l'autre ensemble est utilisé pour le test. Les deux modes (normal/test) fonctionnent avec deux horloges séparées. Entre la préparation des données de test et la collecte des données en sortie, le circuit fonctionne en mode normal. Alterner les deux modes permet de gagner en temps de test [15]. D'autres types d'architectures de chaîne existent comme la chaîne partielle qui passe à travers quelques registres seulement. Certes la chaîne partielle réduit le temps de test,

puisque les bits sont décalés dans un nombre réduit de registres, mais le fait de trouver les « bons » registres à inclure dans la chaîne nécessite un traitement topologique du circuit, ce qui peut s'avérer coûteux [16]. Une architecture à chaînes multiples [17] peut aussi être adoptée où les registres sont mis dans des chaînes séparées. Chacune possède sa propre broche d'entrée et sa propre broche de sortie. Comparées à la chaîne sérielle élémentaire, les chaînes en parallèle réduisent le temps de test, mais elles nécessitent des broches de test supplémentaires. Dans le cas où les registres ne sont pas de même taille, les chaînes sont dites indépendantes et chacune devrait avoir son propre signal d'horloge.

D'autres architectures de chaînes existent comme la chaîne « ILLINOIS » [18] appelée aussi « Broadcast ». L'idée de base de cette architecture est que dans un circuit donné, plusieurs sous modules sont indépendants les uns des autres, donc ils peuvent être testés parallèlement (simultanément) par le même vecteur de test. La figure 1-5 illustre la chaîne de balayage de type ILLINOIS. Le principe est de diviser une longue chaîne de balayage sérielle en plusieurs chaînes en gardant la même broche de balayage. Les données provenant des différentes chaînes de balayage sont récupérées dans un MISR (*Multiple Input Signature Register*) pour être compressées. Le fait de connecter plusieurs modules à la même broche de balayage permet de réduire le nombre de broches utilisées. L'architecture ILLINOIS fonctionne en deux modes : Mode « *Broadcast* » et mode sériel. Dans le mode « *Broadcast* » quelques pannes logiques sont non-testables, le mode sériel est alors utilisé pour avoir une couverture quasi totale de toutes les pannes. Une telle architecture réduit le nombre de broches nécessaires pour le test ainsi que la quantité de mémoire requise pour stocker les données. Le temps de test est également réduit par un facteur N (N étant le nombre de chaînes).

Une autre architecture de chaîne de balayage porte le nom de « *Random Access Scan* (RAS) ». C'est aussi une alternative à la chaîne de balayage sérielle. L'idée de base est d'associer un décodeur d'adresses au module sous test [19]. Le décodeur sélectionne le registre à tester. Une architecture plus avancée du RAS a aussi été proposée [20]. Il s'agit de créer une matrice de registres et à l'aide d'un décodeur colonne et d'un décodeur ligne, un registre en particulier est sélectionné pour le test.

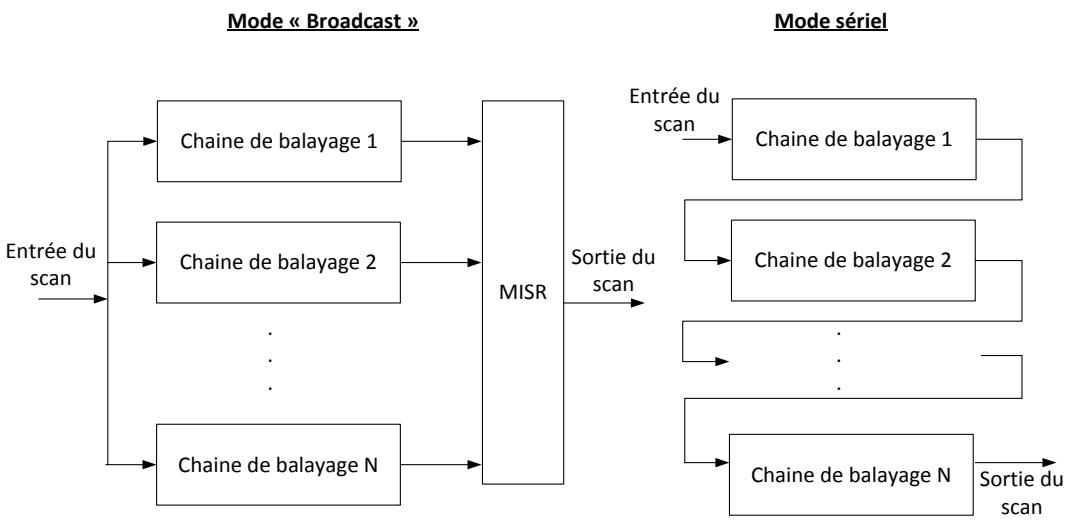


Figure 1-5: Chaîne ILLINOIS.

Une telle architecture permet d'accélérer le temps de test et réduit considérablement le volume nécessaire pour stocker les données ainsi que la puissance consommée. Cependant, elle nécessite des ressources matérielles supplémentaires pour créer la matrice de registres.

1.4.3 Tolérance aux pannes des systèmes JTAG

La chaîne de balayage qui est la base de ce projet de maîtrise est une chaîne JTAG reconfigurable et tolérante aux pannes. Dans ce paragraphe, un bref bilan des techniques de tolérance aux pannes au niveau des chaînes JTAG est dressé.

Problème avec une seule chaîne sérielle

Dans l'architecture JTAG de base appelée aussi architecture en anneau (*single ring architecture*), des données de test sont envoyées à partir du contrôleur de test (port TDI) pour être décalées à travers toutes les unités sous test (UUT) du circuit. La sortie d'une unité est connectée à l'entrée de l'unité suivante de sorte à former une seule chaîne. La dernière unité renvoie les données au contrôleur de test (port TDO). La figure 1-6 schématise l'architecture en anneau avec un signal TMS partagé. En effet, toutes les unités sous test partagent le même signal de contrôle TMS, c'est-à-dire que toutes les UUT exécutent les mêmes opérations (*Shift*, *capture*, *update*..). Cependant, les instructions effectuées peuvent être différentes. Une architecture en anneau avec des signaux TMS séparés est aussi possible et permet de tester chaque unité séparément.

L'avantage de l'architecture en anneau est qu'elle est simple à implémenter puisqu'elle ne nécessite aucun matériel supplémentaire. Deux inconvénients majeurs de l'architecture en anneau sont à mentionner :

1) Pas de tolérance aux pannes : Il suffit d'avoir une seule unité sous test défectueuse et toute la chaîne de balayage est brisée.

2) Dans le cas d'un système complexe à plusieurs cartes, une telle chaîne de balayage devrait parcourir toutes les puces de chaque carte ce qui risque d'allonger le temps de test.

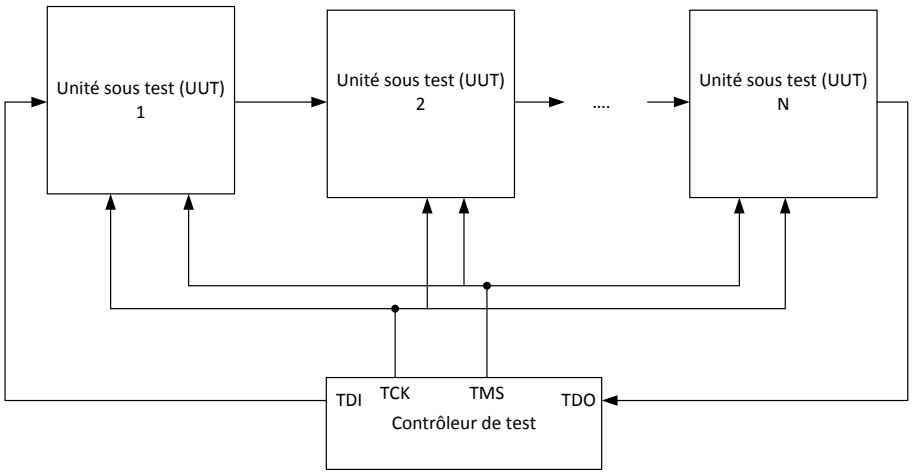


Figure 1-6 Architecture JTAG en anneau.

Alternatives à l'architecture en anneau

- Architecture à plusieurs anneaux (étoile)

Dans cette architecture (Figure 1-7), chaque UUT possède son propre contrôleur de test, c'est-à-dire ses propres signaux de données et de contrôle. Comme si chaque UUT est isolée. Une telle architecture rend le test plus rapide vu que toutes les UUT peuvent être testées en parallèle. Cependant puisque chaque UUT nécessite un contrôleur de test avec 4 (ou 5) signaux chacun, pour un système complexe, le nombre de ports de test risque d'être trop élevé, ce qui peut rendre une telle architecture inefficace.

- Architecture multi-drop

Cette architecture permet de pallier les problèmes rencontrés avec les deux architectures précédentes (anneau et étoile). Un seul contrôleur de test est utilisé pour réduire le nombre de ports dédiés pour le test (voir figure 1-8). Des opérations sont effectuées sur chaque UUT une à la fois. Pour sélectionner l'UUT à tester, chaque unité est dotée d'une porte adressable. L'inconvénient de cette architecture est qu'elle nécessite l'implémentation d'une porte adressable dans chaque unité sous test.

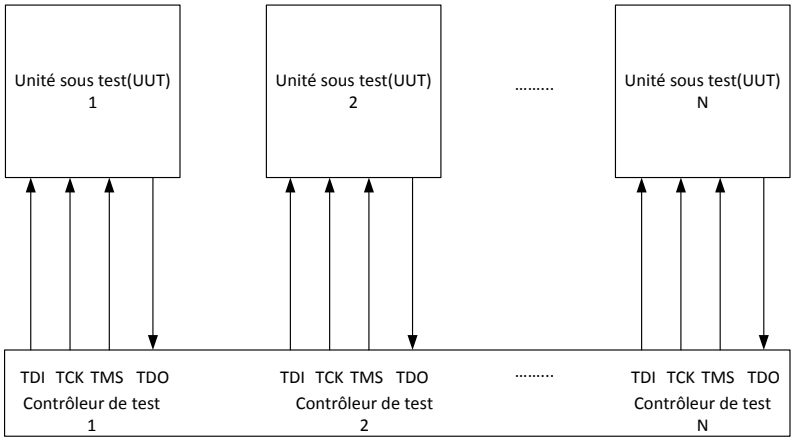


Figure 1-7: Architecture JTAG en étoile.

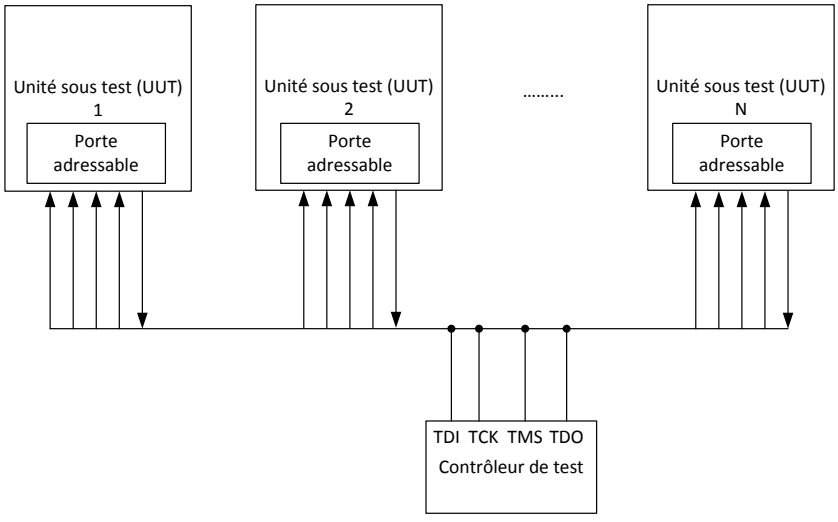


Figure 1-8: Architecture JTAG Multi Drop.

1.5 Architecture de la chaîne JTAG adoptée pour le WaferIC

Étant donné que l'objectif principal est de diagnostiquer la chaîne JTAG utilisée pour la configuration ou le test des cellules du WaferIC, une présentation de l'architecture interne des cellules ainsi que de l'interconnexion entre les cellules est nécessaire.

1.5.1 Structure interne d'une cellule unitaire

Le modèle du WaferIC, tel que proposé dans [21], est une matrice de cellules toutes identiques au niveau physique. Elles ne sont différenciées que par le contenu de leurs mémoires internes. Ces cellules sont configurées conformément au protocole JTAG et en utilisant des chaînes de balayage reconfigurables et tolérantes aux pannes proposées par Yan Basile-Bellavance [22] en 2009. Comme le montre la figure 1-9, chaque cellule contient un contrôleur TAP avec ses registres associés qui sont le registre d'instruction, de données et de bypass (les registres d'instruction et de bypass ne sont pas montrés dans la figure) et elle contient aussi un cœur logique (*Cell logic core*). Deux registres de données supplémentaires Freg et Creg sont ajoutés à la cellule. Le registre Freg permet d'obtenir une chaîne de balayage reconfigurable. Il permet de sélectionner la cellule voisine (Nord, Est, Ouest ou Sud) vers laquelle le flux de données sortant TDO sera dirigé à travers le démultiplexeur DEMUX. Dans cet exemple, il y a un lien TDO vers chaque cellule voisine dans les quatre directions, mais il pourrait y avoir plus ou moins de liens selon le niveau de tolérance souhaité. Le registre de données Creg est ajouté pour configurer le noyau logique de la cellule. À l'entrée de la cellule, une porte logique OU permet de sélectionner le flux de données provenant de la cellule précédente (Nord, Est, Ouest ou Sud). Les sorties du démultiplexeur sont mises à zéro quand le Freg est à zéro ou lors du reset. Ceci garantit que la porte logique OU ne reçoive aucune donnée quand le lien de sortie n'est pas sélectionné.

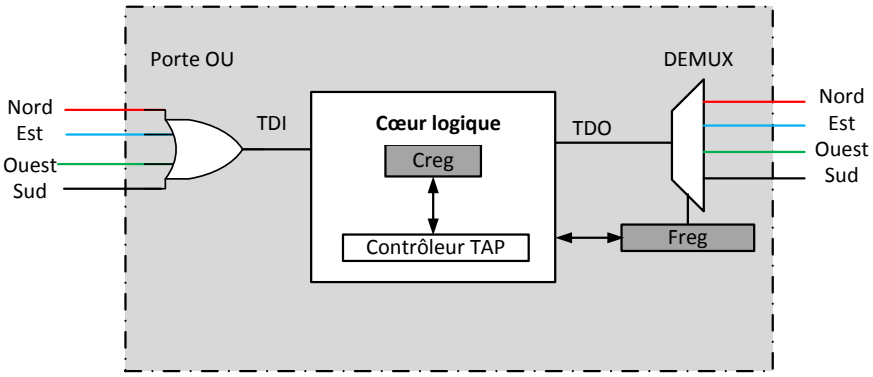


Figure 1-9: Architecture interne d'une cellule.

En dehors du registre de données Freg, les registres associés au contrôleur TAP sont :

- Le registre d'instruction : C'est un registre à 5 bits qui permet d'activer le mode désiré pour la cellule moyennant le décodeur d'instructions.
- Le registre Bypass : C'est un registre à un seul bit. Il permet de passer directement à la cellule suivante dans la chaîne sans effectuer aucune opération.

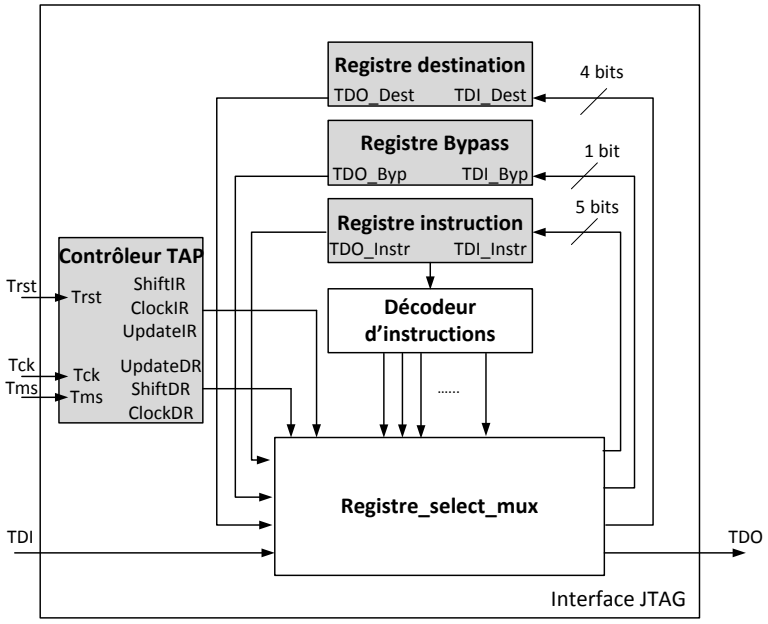


Figure 1-10: Interface JTAG.

La figure 1-10 illustre le fonctionnement du contrôleur TAP comme on le trouve dans la norme JTAG [1]. Le contrôleur TAP est une machine à états finis comportant seize états et commandée par le signal externe TMS provenant du port JTAG. Le TAP peut être divisé en trois sections :

Une section sert à la réinitialisation et les deux autres sections sont dupliquées sauf que l'une concerne le registre d'instruction et l'autre concerne les registres de données. Le passage d'un état à un autre se fait lors du front montant de l'horloge TCK. Le contrôleur TAP décide à travers le *register_select_mux* de remplir soit le registre d'instruction, soit le registre de destination (Freg) ou bien le registre Bypass. Par exemple, si le registre d'instruction est rempli avec l'instruction désirée durant l'état « shift IR », le contrôleur TAP sauvegarde l'instruction en passant par l'état « update IR » et le décodeur d'instruction signale au *register_select_mux* d'entrer dans le mode sélectionné par l'instruction. Ainsi toutes les sorties dépendront du mode entré. Le TAP sert à contrôler le décalage des données dans les registres de scan.

1.5.2 Une chaîne reconfigurable et tolérante aux pannes

Le circuit intégré à très grande échelle, le WaferIC, adopté dans ce projet est composé d'une matrice de cellules ayant toutes une architecture identique à celle présentée dans la figure 1-9, seuls les contenus des mémoires internes diffèrent. Ces cellules sont connectées entre elles avec des liens intercellulaires comme le montre la figure 1-11. Une cellule peut être connectée à plusieurs autres cellules par les liens TDI-TDO. Pour simplifier, on considère qu'il y a un lien TDO pour chaque cellule immédiatement voisine dans les quatre directions (Est, Ouest, Sud et Nord) et un lien TDI provenant de chaque cellule immédiatement voisine dans ces quatre directions. Mais il pourrait y avoir plus ou moins de liens selon le niveau de tolérance souhaité. Parmi les 1024 cellules du réticule, seules deux sont connectées physiquement aux ports externes TDI et TDO.

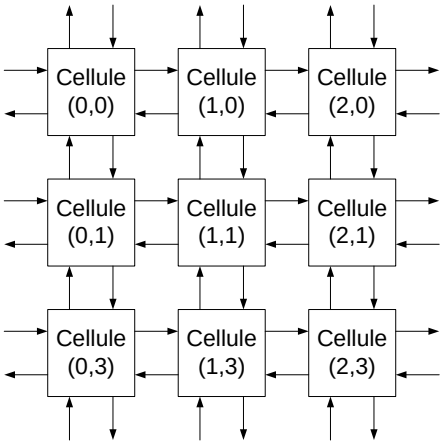


Figure 1-11: Réseau d'interconnexions simplifié dans une matrice de 3×3.

Les cellules du réticule sont configurées depuis l'extérieur avec le protocole JTAG. En effet, pour configurer une cellule particulière, il faut créer un chemin à travers lequel les données de configuration seront envoyées vers cette cellule. Le chemin commence par la cellule liée physiquement au port TDI, parcourt un certain nombre de cellules jusqu'à arriver à la cellule à configurer, puis ressort par la cellule TDO. N'importe quel chemin commençant par TDI et finissant par TDO et qui ne visite aucune cellule deux fois peut être crée. La figure 1-12 montre deux exemples de chemins.

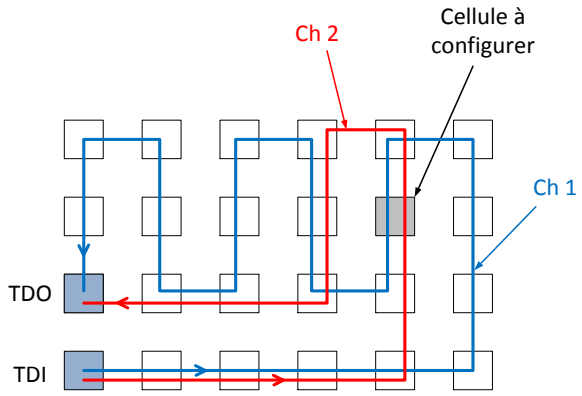


Figure 1-12: Exemple de chemins au sein d'une structure multicellulaire.

Si tous les éléments du chemin comme les liens et les contrôleurs TAP des cellules ne contiennent aucune défectuosité, alors n'importe quel flux de données envoyé à travers ce chemin fonctionnel devrait être observé dans le port de sortie TDO.

L'architecture proposée à la figure 1-9 ne permet pas de traverser la même cellule plus qu'une seule fois. Cette architecture est dite unidirectionnelle. En effet, la cellule ne possède qu'un seul registre de destination, Freg ce qui fait que la chaîne de balayage ne peut pas revenir sur elle-même. Cependant, il est possible de rendre l'architecture bidirectionnelle et dans ce cas la chaîne de balayage peut traverser la même cellule deux fois. Cela se fait au détriment de l'ajout de ressources matérielles comme l'ajout d'un registre de données Freg supplémentaire ainsi que le démultiplexeur associé. Il faut également ajouter une deuxième porte logique OU à l'entrée de la cellule pour recevoir le bon flux de bits TDI ainsi qu'un autre registre Bypass et les interconnexions nécessaires. La figure 1-13 illustre l'architecture interne de la cellule qui permet d'avoir une chaîne intercellulaire bidirectionnelle.

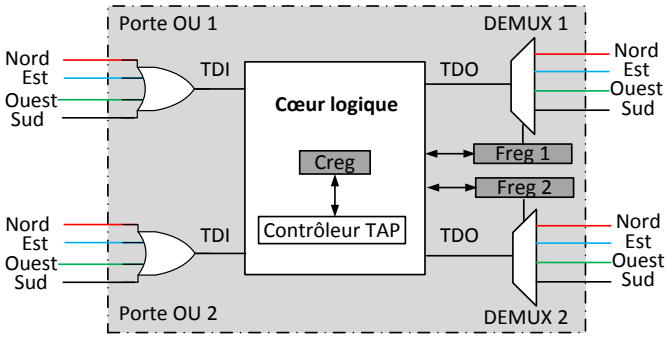


Figure 1-13: Architecture interne d'une cellule qui permet d'avoir une chaîne JTAG bidirectionnelle.

Certes la bidirectionnalité apporte un niveau de tolérance aux pannes plus élevé, mais elle nécessite également beaucoup plus de ressources matérielles. L'architecture bidirectionnelle n'a pas été retenue parce que les coûts additionnels promettaient de dépasser les bénéfices attendus. Par conséquent, le WaferIC est donc basé sur l'architecture unidirectionnelle.

Le contrôle externe

L'architecture de la chaîne JTAG adoptée pour le WaferIC est reconfigurable et tolérante aux pannes : chaque cellule est connectée à ses quatre voisines et à d'autres cellules lointaines. Une cellule source reçoit un flux de données en entrée et le redirige vers une cellule destination. Si la cellule destination est défectueuse, ou bien si le lien intercellulaire correspondant est défectueux, alors la cellule source peut toujours rediriger son flux de bits vers une autre cellule destination. Ceci n'est possible que si un diagnostic préalable est fait pour repérer les cellules et les liens défectueux. Un autre aspect de la tolérance aux pannes est le contrôle externe [4] : Si le contrôleur TAP d'une cellule est défectueux, la cellule peut toujours être configurée à travers le contrôleur TAP fonctionnel de sa cellule voisine. La figure 1-14 schématise un exemple d'application du contrôle externe dans une grille de 3×3 cellules extrait de [21].

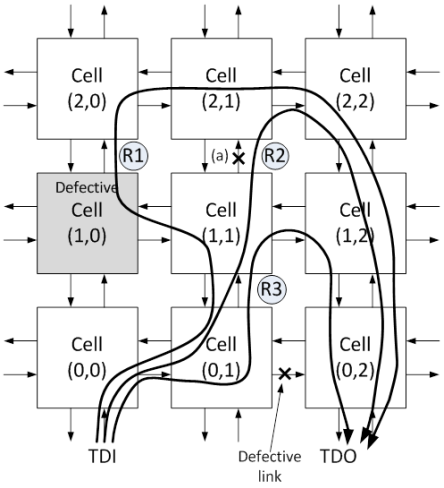


Figure 1-14: Matrice de 3×3 cellules avec des liens (x) et un contrôleur TAP (Cellule en gris) défectueux. Trois chaînes différentes (R1, R2, R3) parcourent la matrice.
Figure extraite de [21].

Considérons trois chaînes distinctes R1, R2 et R3. Chacune commence par la cellule liée physiquement au port TDI. Elle parcourt un certain nombre de cellules et ressort par la cellule connectée physiquement au port TDO. Les défectuosités présentes dans cette grille sont :

- La cellule (1,0).
- Le lien intercellulaire nord sortant de la cellule (1,1).
- Le lien intercellulaire Est sortant de la cellule (0,1).

Ces défectuosités rendent les chemins R1 et R2 non fonctionnels et par conséquent les cellules (1,0), (2,0), (2,1) et (2,2) inutilisables. En implémentant le mécanisme du contrôle externe dans chaque cellule du WaferIC, les défectuosités présentes dans la grille rendent seulement la cellule (2,0) inutilisable. Les cellules (1,0), (2,1) et (2,2) peuvent être configurées et contrôlées respectivement à travers les cellules (0,0), (1,1) et (1,2) puisque le chemin R3 est fonctionnel.

1.5.3 Montage expérimental pour communiquer avec un réticule du WaferIC

Un prototype, nommé mini-WaferIC (Figure 1-15), qui représente le 1/1200^{éme} de la superficie totale d'un circuit intégré à l'échelle de la tranche dont le diamètre est de 200 mm a été conçu par l'équipe DreamWafer^{TM} pour effectuer les tests. Le mini-WaferIC est une version miniaturisée du WaferIC suffisante pour émuler le comportement de base du circuit sans avoir à produire le

WaferIC au complet (ce qui aurait été dispendieux et fort risqué). Le mini-WaferIC contient seulement 4 réticules, chacun ayant 32×32 cellules. Les réticules sont connectés entre eux en utilisant la technique « *inter-reticle stitching* » qui consiste à relier les réticules voisins par une couche de métal définie à partir des masques de photolithographie. Cinq signaux de contrôle JTAG permettent la programmation du mini-WaferIC. Le paragraphe suivant décrit le montage fait pour pouvoir communiquer avec le mini-WaferIC.

Montage expérimental pour communiquer avec le mini-WaferIC

La figure 1-16 illustre le montage expérimental fait afin d'entrer en communication avec un réticule du mini-WaferIC. Les flux de bits JTAG sont injectés dans un réticule moyennant un FPGA AGL060 de la famille IGLOO (Actel). Le FPGA contient jusqu'à 144 kbits de mémoire SRAM et jusqu'à 300 ports d'entrée/sortie. Le FPGA est embarqué dans un module d'alimentation (Power Block) qui lui fournit un support mécanique ainsi que l'alimentation en puissance. Le FPGA est programmé en utilisant le programmeur FlashPro4 d'Actel. Une fois que le FPGA est programmé, les flux de bits JTAG, générés avec un code Matlab, sont transmis au FPGA à travers le convertisseur UM232R. Finalement, le FPGA transmet les flux de bits au réticule. La programmation du FPGA ainsi que la génération du flux de bits JTAG avec Matlab ont été faits par un co-équipier, Gontran Sion, étudiant à la maîtrise.

Alimentations en tension

Quatre alimentations sont requises :

- 1.5 V : Pour alimenter le cœur du FPGA.
- 3.3 V : Pour alimenter les broches du FPGA.
- 12 V : Pour alimenter le régulateur du Power Block. Celui-ci fournit une tension de 3.3 V au mini-WaferIC.
- 1.8V : Pour alimenter les quatre réticules.

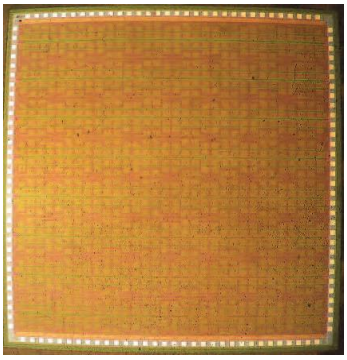


Figure 1-15: Prototype du WaferIC (mini-WaferIC) réalisé par l'équipe DreamWafer^{TM}.

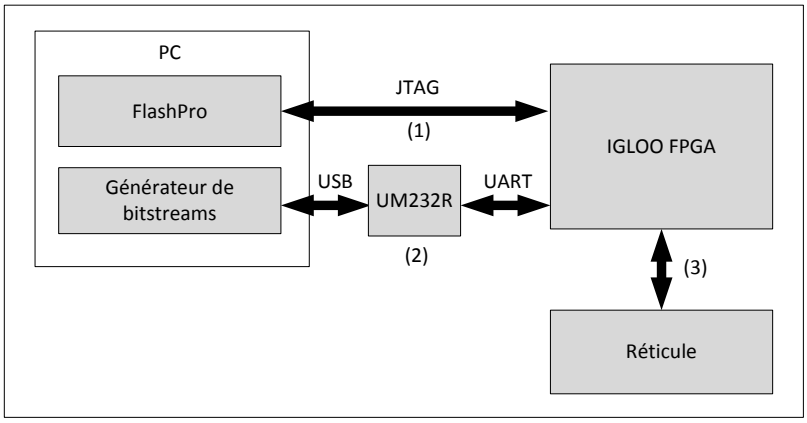


Figure 1-16: L'environnement de test du mini-WaferIC réalisé par l'équipe.

1.6 Problématique de ce mémoire

Les cellules du WaferIC sont configurées en utilisant le protocole de test JTAG. Un flux de données est envoyé à travers la cellule connectée physiquement au port TDI, il parcourt un certain nombre de cellules puis ressort par la cellule connectée physiquement au port TDO. Ce flux de données crée à l'intérieur du WaferIC une chaîne de balayage reconfigurable. Le terme reconfigurable signifie que les interconnexions entre les cellules peuvent être reprogrammées. Un problème est que si la chaîne de balayage rencontre ne serait-ce qu'un seul élément défectueux, elle devient complétement inutilisable. L'idée est de rendre cette chaîne capable de passer seulement par des éléments fonctionnels. Pour y parvenir, un diagnostic doit être fait au préalable pour localiser et identifier le plus d'éléments fonctionnels possibles.

La contribution de ce projet de maîtrise consiste à développer des algorithmes de diagnostic de la chaîne JTAG reconfigurable. Ces algorithmes ont pour but de repérer le plus de cellules et de liens fonctionnels possible dans le WaferIC. La contribution de ce mémoire se situe dans la première étape du flux de données du WaferConnect qui est l'étape de diagnostic.

Méthodes de diagnostic

La chaîne JTAG à tête chercheuse, a fait l'objet d'un brevet [23] .Vu l'unicité du projet et l'originalité de la chaîne JTAG à tête chercheuse, comparer nos travaux à des travaux antérieurs n'est pas faisable. Toutefois, quelques travaux tenant sur le diagnostic des chaînes de balayage seront cités.

Plusieurs méthodes existent dans la littérature pour diagnostiquer les chaînes de balayage. On y trouve des méthodes basées sur l'ajout de matériel (*hardware*) et d'autres basées sur la simulation. Dans [24] du matériel a été ajoutée pour connecter les registres de décalage en paires de telle sorte qu'un registre peut diagnostiquer l'autre. Les registres mis en paires portent le nom de « partenaires ». L'inconvénient majeur de cette méthode est que si les deux partenaires sont défectueux, aucun diagnostic ne pourrait être fait. Dans [25], de la circuiterie est ajoutée aux registres de scan pour activer et désactiver (*set and reset*) leurs ports de sortie. Cette technique permet de détecter une erreur singulière de type collage (*Stuck at fault*). Dans [26], une porte logique XOR est insérée entre deux registres de balayage consécutifs. Un signal supplémentaire est utilisé pour contrôler tous les registres de balayage. La chaîne de balayage est reconfigurée en utilisant ces portes logiques XOR pour détecter les erreurs de collage. Dans [27], on trouve une méthode pour diagnostiquer la chaîne de balayage en utilisant la technique de segmentation. En effet, la détection des défectuosités se fait en segmentant et en reconfigurant statiquement et dynamiquement les chaînes. Toutes les chaînes sont divisées en un nombre égal de segments. Un multiplexeur est placé entre deux segments consécutifs de la même chaîne. Le multiplexeur permet de connecter un segment donné soit au prochain segment de la même chaîne ou bien au prochain segment de la chaîne immédiatement voisine. En contrôlant les multiplexeurs pour interconnecter les segments de différentes façons, les chaînes peuvent être reconfigurées et les segments défectueux seront ainsi repérés. La condition à satisfaire pour pouvoir utiliser cette méthode est que les chaînes soient toutes de même longueur et qu'elles comportent toutes le même nombre de segments (les segments doivent être de même taille). D'autres techniques de

diagnostic au niveau logique existent. Ces techniques peuvent bien être appliquées pour diagnostiquer les chaînes de balayage. Le livre d'Abramovici, A.Breuer et D.Friedman [28] en résume quelques-unes. On y trouve la technique nommée « *fault dictionary* » qui consiste à comparer la réponse obtenue de la partie sous test à d'autres réponses calculées au préalable. L'inconvénient de cette technique de diagnostic est qu'elle nécessite un temps de calcul assez long, ainsi qu'une large mémoire pour calculer et stocker toutes les réponses possibles de la partie sous test. Pour diagnostiquer un circuit au niveau logique une technique nommée « *Guided-probe testing* » peut aussi être utilisée. Il s'agit de retracer l'erreur depuis la porte de sortie où elle a été détectée jusqu'à arriver à sa source. Le test se fait sur différentes lignes en utilisant une sonde. Déplacer la sonde d'une ligne à une autre prend beaucoup de temps c'est pour cela que quelques techniques permettant de diminuer le nombre de lignes à tester par la sonde ont vu le jour [29]. Une troisième technique de diagnostic est la technique de réduction de la partie sous test « *Diagnosis by UUT reduction* ». Il s'agit de tester la moitié du circuit tout en désactivant l'autre moitié. Si la partie testée est trouvée fonctionnelle alors la partie qui a été désactivée contient la défectuosité et elle devient à son tour la partie à tester. Finalement, des systèmes experts basés sur l'intelligence artificielle peuvent aussi être utilisés pour le diagnostic. Ces systèmes imitent la capabilité d'un être humain à résoudre un problème particulier [30]. Ils se basent sur des règles de type : Si (condition) alors (action(s)). Ces systèmes sophistiqués sont peu utilisés car ils demandent un niveau de programmation très élevé et leur réalisation est assez coûteuse.

Coût du diagnostic

Durant la phase de fabrication d'un système, le test se fait du bas niveau vers le haut niveau. Par exemple en commençant par le circuit intégré jusqu'à arriver au PCB. Ceci assure l'obtention d'un système composé uniquement d'éléments fonctionnels, ce qui réduit considérablement le temps et le coût du diagnostic au cas où le système s'avère défaillant. Par exemple, on cite souvent que la détection d'une défectuosité sur un circuit intégré coûte de l'ordre de 1$. Le coût pour localiser ce même circuit défectueux monté sur une carte PCB s'élève à 10$. Quand la carte défectueuse est montée sur un système, le coût de localisation du circuit intégré défaillant peut s'élever jusqu'à 100$. Cet exemple montre à quel point le processus de diagnostic est coûteux.

Repérer les éléments fonctionnels

Diagnostiquer un circuit a généralement pour but de localiser ou identifier les éléments défectueux afin de les remplacer par des éléments fonctionnels. En effet, dans le cadre de la tolérance aux pannes, la majorité des circuits contiennent des éléments nécessaires au fonctionnement normal du circuit et d'autres éléments utilisés comme rechange dans le cas où il y a un module défectueux. Cette technique de tolérance aux pannes est appelée la technique de redondance. On la trouve dans les mémoires par exemple. Diagnostiquer une mémoire revient à localiser ses éléments défectueux afin de pouvoir les remplacer par des éléments fonctionnels. Dans le cas de la technologie WaferIC, l'approche prise pour le diagnostic est de repérer le maximum d'éléments fonctionnels et ceci entre autres pour deux raisons :

- Une chaîne JTAG doit passer seulement par des éléments caractérisés comme fonctionnels pour éviter d'être brisée.
- Avec le mécanisme du contrôle externe, une cellule dont la circuiterie interne est défaillante peut toujours être configurée à partir d'une cellule immédiatement voisine fonctionnelle.

Pour ces deux raisons, on cherche à collecter le plus d'éléments fonctionnels possible pour configurer toutes les cellules du WaferIC.

1.7 Contributions de ce mémoire

La principale contribution de ce mémoire est le développement de méthodes algorithmiques pour diagnostiquer la chaîne JTAG utilisée pour la configuration et le test du WaferIC. Le but du diagnostic est de trouver le plus d'éléments (cellules et liens) fonctionnels au sein d'un réticule du WaferIC. L'idée est de créer plusieurs chemins à l'intérieur du réticule et d'observer la sortie au niveau de chacun. Si la sortie correspond à ce qui est attendu, alors tous les éléments du chemin sont fonctionnels. Dans le cas contraire, un ou plusieurs éléments défectueux sont présents dans le chemin. Dans ce cas, un algorithme de recherche basé sur le principe de la dichotomie est alors appliqué sur le chemin trouvé non fonctionnel pour localiser la ou les défectuosité(s). Dans le cas où l'algorithme de dichotomie n'arrive pas à localiser étroitement la défectuosité, deux algorithmes heuristiques ont été développés pour améliorer la résolution du diagnostic et discerner le plus étroitement possible la ou les défectuosités. Ces algorithmes de

recherche ont pour but d'élargir l'espace des éléments fonctionnels du réticule. Les algorithmes de diagnostic développés ont été testés sur un prototype à petite échelle du WaferIC disponible au laboratoire. Plusieurs défectuosités groupées y ont été détectées et localisées. Des résultats de simulation ont également été dressés montrant le comportement des algorithmes en présence d'un ou de plusieurs élément(s) défectueux. D'autres contributions d'ordre théorique de ce mémoire sont à mentionner. Tout d'abord il y a l'analyse de complexité en temps de diagnostic qui permet de choisir les chemins les plus adéquats à créer au sein du réticule. Il a été démontré dans ce mémoire que la complexité temporelle pour établir un chemin de N cellules croit en $O(N^2)$. Ensuite, une comparaison entre deux couvertures possibles des éléments du réticule est dressée : couverture par des chemins rectangulaires et couverture par des chemins en serpentin. Cette comparaison a conduit à choisir des chemins rectangulaires pour couvrir les cellules et les liens du réticule car ils nécessitent un temps de configuration beaucoup moins élevé (en se basant sur l'analyse de complexité établie). Une autre contribution d'utilité pratique est la modélisation du réticule et des chemins rectangulaires par le logiciel Matlab afin de déterminer la taille des différents chemins rectangulaires et de déterminer ainsi la taille des flux de bits nécessaires pour établir ces chemins.

1.8 Sommaire

Ce chapitre a présenté en premier lieu quelques limites des systèmes sur PCB. Ces limites ont conduit à chercher plusieurs autres alternatives parmi lesquelles figurent les systèmes intégrés au niveau de la tranche de Silicium. Dans le cadre de ces systèmes, le brevet de Richard Norman a été déposé en 2006. Il a présenté un nouveau type de circuit intégré qui est le WaferIC. Cette technologie brevetée est la pièce maîtresse de plateforme de prototypage rapide des systèmes numériques, WaferBoard^{TM}, qui a aussi été introduite dans ce chapitre. Le WaferIC a fait l'objet de plusieurs sujets de thèse et de maîtrise tout au long de ces huit dernières années, dont ce présent sujet. Ce projet de maîtrise propose des méthodes algorithmiques pour diagnostiquer la chaîne JTAG nécessaire pour configurer le WaferIC. Cette chaîne JTAG, décrite dans ce chapitre, présente une architecture originale qui a été conçue en 2009 et qui vient d'être publiée en 2014. C'est une architecture reconfigurable et tolérante aux pannes. Pour introduire cette architecture, il a fallu d'abord faire référence dans ce chapitre à quelques architectures de chaînes de balayage existantes. Ce chapitre a également fait un survol de quelques techniques de

diagnostic de la chaîne de balayage qu'on peut trouver dans la littérature. Finalement, la problématique a été posée et les contributions de ce mémoire ont été soulignées.

CHAPITRE 2 OUTILS NÉCESSAIRES AU DIAGNOSTIC DE LA CHAÎNE DE BALAYAGE

2.1 Introduction

Le cœur de la plateforme WaferBoard^{TM} est le WaferIC. C'est un circuit électronique de type « *Wafer Scale Integration* » c'est-à-dire un circuit intégré à l'échelle de la tranche. C'est un substrat programmable sur lequel seront déposés les puces et les circuits à interconnecter. Il envoie les informations sur le placement des circuits au logiciel WaferConnect et il reçoit et exécute une « *netlist* » provenant de l'utilisateur contenant les interconnexions à faire. Ce substrat intelligent est composé de 76 réticules. Chaque réticule est une matrice de 32×32 cellules, soit 1024 cellules connectées entre elles avec des liens intercellulaires. Les cellules du WaferIC sont testées et configurées avec des chaînes de balayage essentiellement conformes à un protocole JTAG étendu. Une chaîne commence par la cellule liée physiquement au port TDI, parcourt un certain nombre de cellules, puis ressort par la cellule liée physiquement au port TDO. Utiliser une seule chaîne sérielle, comme c'est normalement le cas avec les PCB, pose un problème. En effet, si la chaîne rencontre ne serait-ce qu'une seule défectuosité, elle sera complètement brisée et par conséquent inutilisable. Une défectuosité peut se trouver au niveau d'un lien intercellulaire ou bien au niveau de la circuiterie interne d'une cellule comme le contrôleur TAP par exemple ou dans l'un de ses registres. Pour cette raison, possiblement avant chaque utilisation de la plateforme, un diagnostic doit être fait pour savoir quelles ressources utiliser. Dans un des modes de fonctionnement envisagé, le diagnostic doit se faire automatiquement au démarrage de la plateforme. Il sert à repérer les ressources fonctionnelles et les ressources dont la fonctionnalité n'est pas garantie. Le diagnostic permettra de ne pas utiliser des ressources défectueuses (ou probablement défectueuses) pour éviter des pertes de temps pendant le déverminage du système. L'objectif de ce projet de maîtrise est d'élaborer des algorithmes de diagnostic de la chaîne JTAG du WaferIC.

Ce chapitre se divise en quatre sections comme suit : La première section présente des cas de figures de défectuosités. La seconde section donne quelques définitions utiles pour le reste du mémoire. La troisième section est une analyse de complexité temporelle de configuration d'un chemin. La quatrième section est une comparaison entre deux couvertures possibles du réticule :

la couverture par des chemins en rectangles et la couverture par des chemins en serpentins. Finalement, la cinquième section décrit en détail la couverture par des chemins rectangulaires, et ce en modélisant sur MATLAB ces chemins dans un réticule de taille 32×32.

2.2 Quelques exemples de défectuosités

Étant donné que l'objectif principal est de diagnostiquer la chaîne de balayage, une présentation de quelques cas possibles de défectuosités est nécessaire. Une défectuosité peut toucher aussi bien un lien intercellulaire que la circuiterie interne d'une cellule.

2.2.1 Lien intercellulaire défectueux

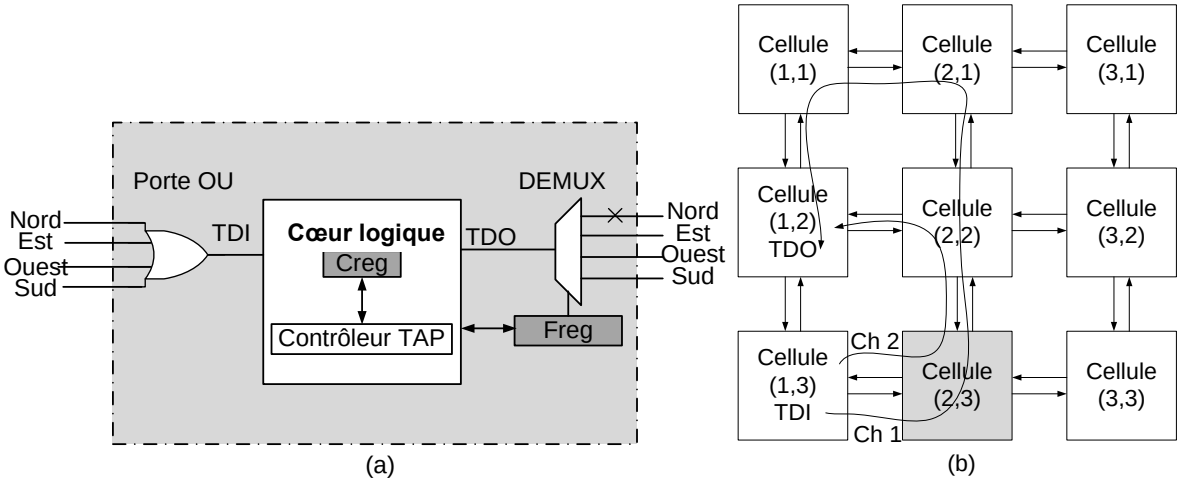


Figure 2-1: Défectuosité au niveau d'un lien intercellulaire (a) Architecture interne de la cellule (2,3) (b) matrice de 3×3 cellules.

Étant donné l'exemple de la figure 2-1. Le lien intercellulaire nord de la cellule de coordonnées (2,3) est défectueux. Ceci engendre que n'importe quel chemin (exemple Ch1 et Ch2) passant par ce lien est un chemin non fonctionnel. En effet, ce chemin ne peut pas transmettre les données de configuration depuis la cellule de départ TDI jusqu'à la cellule d'arrivée TDO. Les cellules ayant un lien défectueux peuvent toutefois être configurées en adoptant d'autres chemins. Par exemple la cellule de coordonnées (2,3) peut être configurée ou testée avec un chemin passant par les cellules (1,3), (2,3), (3,3), (3,2), (2,2), et (1,2). Dans certains cas, un lien intercellulaire défectueux rend la configuration de toute la cellule impossible. Par exemple si le lien Ouest sortant de la cellule (3,1) est défectueux, cette cellule ne pourra pas être configurée ou testée par

une chaîne de balayage (unidirectionnelle) vu qu'il n'y aura aucun chemin fonctionnel passant par elle. Toutefois, la cellule (3,1) peut être configurée à l'aide du mécanisme de contrôle externe si l'une de ses voisines immédiates est fonctionnelle.

Identification des liens intercellulaires « non diagnosticables » :

L'utilisation d'une chaîne de balayage vient avec deux principales contraintes :

1. Une chaîne de balayage doit être un chemin complet, c'est-à-dire qu'il commence toujours par la cellule TDI et se termine par la cellule TDO.

2. Une cellule ne doit pas être visitée plus d'une seule fois par la même chaîne, puisque l'architecture unidirectionnelle a été adoptée lors de la conception du WaferIC.

Idéalement, tous les liens intercellulaires doivent être visités par des chemins. Sauf que cela dépend systématiquement des positions des ports TDI et TDO. Dans le cas spécifique d'un réticule du WaferIC, les cellules d'entrée et de sortie TDI et TDO se situent dans le coin inférieur gauche, précisément les coordonnées de la cellule TDI sont (0,31) et ceux de la cellule TDO sont (0,30). Cet emplacement particulier des cellules TDI et TDO engendre que parfois il est impossible de trouver un chemin parcourant certains liens tout en respectant les contraintes 1 et 2. Le terme « non diagnosticable » est utilisé pour ces liens. La figure 2-2, montre les liens « non diagnosticables » dans une grille simplifiée de 6×6 cellules. Un réticule de 32×32 cellules contient 3966 liens intercellulaires. 3843 de ces liens sont « diagnosticables » et 123 ne le sont pas. Vu que ce qui nous intéresse ce sont les liens « diagnosticables », pour alléger l'écriture, dans le reste du mémoire le terme lien sous-entend un lien « diagnosticable ».

2.2.2 Défectuosité au niveau de la circuiterie interne de la cellule

Conformément à la norme JTAG, trois registres sont associés au contrôleur TAP : Le registre d'instruction, le registre de destination et le registre bypass. Le contrôleur TAP décide lequel de ces registres sera rempli, il sert ainsi à contrôler le décalage des données dans les registres de scan. Une défectuosité au niveau du contrôleur TAP fait de la cellule une cellule bloquante. Les quatre liens entrants ainsi que les quatre liens sortants d'une cellule bloquante sont inutilisables et par conséquent n'importe quel chemin passant par cette cellule sera incapable de transmettre les flux de données nécessaires au test ou à la configuration des cellules. Soit l'exemple de la figure

2-3. Supposons que le contrôleur TAP de la cellule (2,3) est défectueux. Les cellules (3,3), (3,2), (3,1), (2,1), (2,2) et (1,1) sont inaccessibles car n'importe quel chemin accédant à l'une de ces cellules doit impérativement passer par la cellule bloquante (2,3). Le même problème se présente si une défectuosité touche le démultiplexeur de sortie (DEMUX), son registre de contrôle Freg ou bien la porte logique OU d'entrée de la cellule.

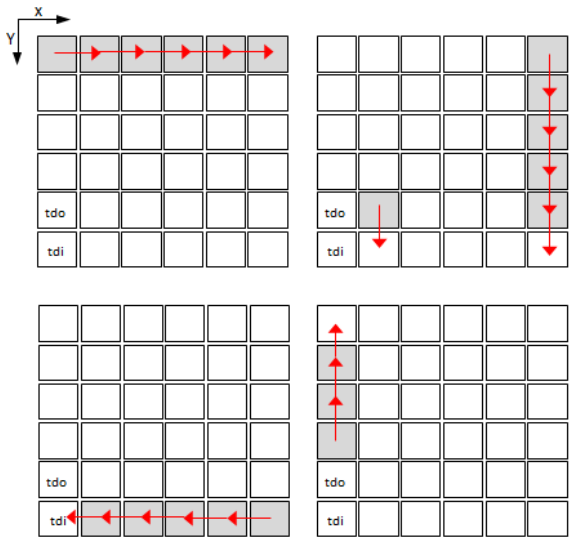


Figure 2-2: Cellules ayant des liens sortants « non diagnosticables » **:** (a) Cellules dont le lien sortant EST est « non diagnosticable », (b) Cellules dont le lien sortant SUD est « non diagnosticable », (c) Cellules dont le lien sortant Ouest est « non diagnosticable », (d) Cellules dont le lien sortant Nord est « non diagnosticable ».

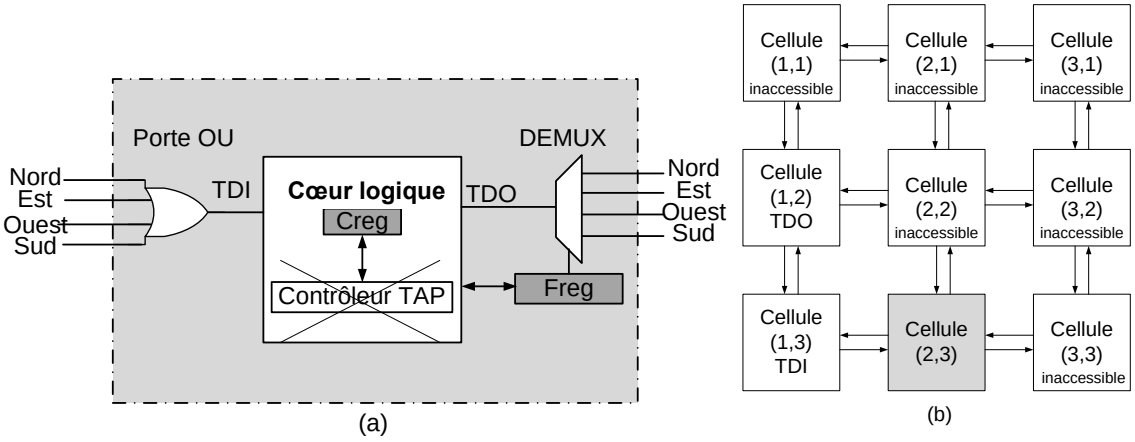


Figure 2-3: Défectuosité au niveau du contrôleur TAP de la cellule (a) Architecture interne de la cellule (2,3) (b) matrice de 3×3 cellules.

2.3 Définition des éléments du réticule

Quelques termes sont à définir pour la suite de ce mémoire :

- Un *élément* peut être un lien intercellulaire, un contrôleur TAP d'une cellule, la broche TDI ou bien la broche TDO. Une chaîne de ces éléments peut créer un chemin commençant du point d'entrée (TDI) et se terminant par le point de sortie (TDO).
- Un *chemin* est constitué d'un ensemble de couples {Cellule, Lien intercellulaire}.
- Un *chemin fonctionnel* est un chemin ne contenant aucun élément défectueux. Il permet la propagation des signaux du port (TDI) jusqu'au port (TDO) sans aucune erreur.
- Un *chemin non fonctionnel* est un chemin qui ne permet pas la propagation du signal du port (TDI) jusqu'au port (TDO) ou bien le signal observé au niveau du port TDO est un signal erroné à cause d'un ou de plusieurs éléments défectueux dans le chemin.
- Un *élément fonctionnel* est un élément pour lequel il a été démontré qu'il ne cause aucune erreur observable sur au moins un chemin fonctionnel.
- Un *élément défectueux* est un élément pour lequel il a été démontré qu'il crée des erreurs observables sur tout chemin le traversant.
- Un élément *potentiellement défectueux* est un élément pour lequel :
  * il a été démontré qu'il crée des erreurs observables sur un ou plusieurs chemin(s) non fonctionnel(s).
  * il n'a pas été démontré qu'il ne cause aucune erreur observable sur au moins un chemin fonctionnel.
- Un *chemin complet* est un chemin dont la première cellule est la cellule TDI et la dernière cellule est la cellule TDO. Autrement, le chemin est dit *chemin incomplet*. Un chemin peut désigner aussi bien un chemin complet qu'un chemin incomplet.

L'objectif du diagnostic est de trouver le plus possible de cellules et de liens fonctionnels comme première étape et ensuite chercher les zones défectueuses.

Les défectuosités dans un circuit intégré à grande surface sont inévitables. Un chemin passant par une défectuosité est un chemin non fonctionnel. La chaîne JTAG devient tolérante aux pannes si un chemin fonctionnel ou une série de chemins fonctionnels peuvent être trouvés pour passer à

travers toutes les cellules fonctionnelles du réticule. Ceci nécessite une étape préalable de diagnostic pour repérer le maximum d'éléments fonctionnels.

Les cellules et les liens intercellulaires sont initialement dans un état inconnu. Le but du diagnostic est de caractériser les éléments de la chaîne JTAG. Caractériser un élément revient à savoir s'il est fonctionnel, défectueux ou probablement défectueux. La caractérisation des cellules et des liens intercellulaires d'un réticule se résume à la caractérisation des liens d'interconnexion de la circuiterie JTAG. À partir des liens, l'état des cellules peut être déduit. En effet, si une cellule possède quatre liens défectueux en entrée et quatre liens défectueux en sortie alors cette cellule est nécessairement défectueuse (bloquante).

Le diagnostic se fait principalement en deux étapes. La première étape consiste à créer une courte chaîne entre les cellules TDI et TDO pour s'assurer de leur fonctionnalité. La deuxième étape consiste à créer plusieurs chemins et observer le signal à la sortie de chacun. Chaque chemin commence par la cellule TDI, parcourt un certain nombre de cellules et finit par la cellule TDO. Comme l'architecture unidirectionnelle a été adoptée lors de la conception du réticule, il faut tenir compte qu'un chemin ne peut pas traverser la même cellule plus qu'une seule fois. Le diagnostic se fait moyennant deux axiomes (A1 et A2) donnés ci-dessous :

Considérons un chemin p_i constitué de N cellules et de N+1 liens.

$p_i = (C, L)$ où $C = \{c_1, c_{2\ldots}c_N\}$ est une séquence ordonnée de cellules et $L = \{l_0, l_1, l_2\ldots l_N\}$ est une séquence ordonnée de liens.

Soit R, $\bar{R}$, Q et $\bar{Q}$ quatre hypothèses définies par :

R : p_i est fonctionnel.

$\bar{R}$: p_i est non fonctionnel.

Q : $\forall$ *element* $\in p_i$, *element* est fonctionnel.

$\bar{Q}$: $\exists!$ *element* $\in p_i$, *element* est défectueux.

A1 : « Si un chemin est fonctionnel alors tous ses éléments sont fonctionnels », A1: $R \Rightarrow$ Q

A2 : « Si un chemin est non fonctionnel alors au moins un de ses éléments est défectueux», A2: $\bar{R} \Rightarrow \bar{Q}$

L'objectif est de trouver un ensemble d'éléments fonctionnels permettant de configurer toutes les cellules du réticule. Ceci réduit considérablement le temps de configuration du réticule. Pour y parvenir, il faut d'abord trouver un ensemble de chemins qui parcourent la totalité des liens tout en minimisant la taille des flux de bits nécessaires à la création de ces chemins. Pour cela, une analyse de complexité a été dressée. Cette analyse montre que la taille d'un flux de bits nécessaire pour établir un chemin de N cellules croit en N^2. Le paragraphe suivant est une preuve mathématique qui démontre que la complexité temporelle de la configuration d'un chemin de N cellules est de $O(N^2)$.

2.4 Complexité temporelle de création d'un chemin

Le WaferIC est un circuit intégré à grande surface (LAIC) implémenté avec une chaîne de balayage reconfigurable et tolérante aux pannes, qui peut être configurée en utilisant le protocole JTAG. Pour configurer une cellule particulière, un flux de bits est envoyé depuis le port externe TDI pour créer un chemin passant par cette cellule. Le flux de bits inclut des instructions et des données nécessaires pour établir le chemin. Le flux de bits est construit tel que décrit dans le diagramme donné dans la figure 2-4.a et il est basé sur les instructions JTAG « *Shift* » et « *Update* ». Quand la machine à états (FSM) du contrôleur TAP est dans l'état « Shift IR » le registre d'instructions de la cellule de départ est rempli pour mettre la cellule dans le mode destination. Quand la FSM est dans l'état « shift DR », le registre Freg de cette même cellule est rempli selon la destination à prendre (Nord, Est, Ouest ou Sud). Ces étapes sont répétées avec toutes les cellules suivantes dans le chemin jusqu'à atteindre la broche TDO.

Pour prouver que la complexité est de $O(N^2)$, il vaut mieux partir d'un exemple puis généraliser. Considérons le chemin à établir de la figure 2-4.b qui commence par la cellule TDI et passe directement à la cellule voisine TDO sans parcourir aucune autre cellule. La cellule TDI est nommée $cell_1$ et la cellule TDO est nommée $cell_2$. Le flux de bits JTAG nécessaire à la création de ce chemin est donné dans le tableau 2.1.

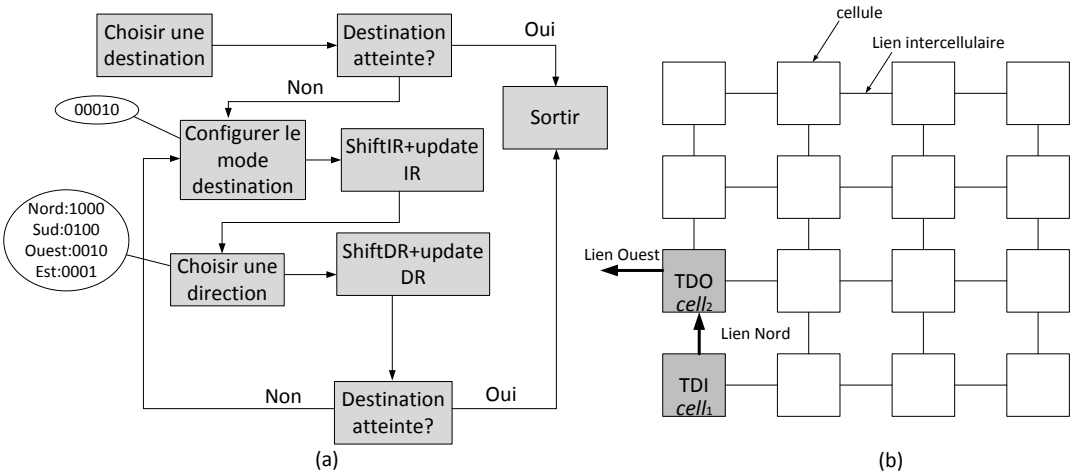


Figure 2-4: Création d'un flux de bits JTAG (a) Organigramme de création d'un flux de bits JTAG (b) Exemple de chemin contenant deux cellules (TDI-TDO).

Tableau 2.1 Flux de données JTAG nécessaire à la création du chemin donné dans la figure 2-4.b.

TMS	État de la FSM	Taille du flux JTAG	Explication	TMS	État de la FSM	Taille du flux JTAG	Explication
0	Reset	6 bits	Atteindre l'état shift_IR	1	exit_dr	6 bits	Atteindre l'état shift_IR
0	idle			0	update_dr		
1	idle			1	idle		
1	select_DR			1	select_DR		
0	select_IR			0	select_IR		
0	capture_IR			0	capture_IR		
0	shift_IR	5 bits	Mettre le registre IR de *cell*$_1$ en mode destination	0	shift_IR	5 bits	Mettre le registre IR de *cell*$_2$ en mode destination
0	shift_IR			0	shift_IR		
0	shift_IR			0	shift_IR		
0	shift_IR			0	shift_IR		
1	shift_IR			1	shift_IR		
1	exit_IR	5 bits	Atteindre l'état shift_DR	1	exit_IR	5 bits	Atteindre l'état shift_DR
0	update_IR			0	update_IR		
1	idle			1	idle		
0	select_DR			0	select_DR		
0	capture_DR			0	capture_DR		
0	shift_DR	4 bits	Remplir $Freg_{cell1}$ Avec la destination Nord	0	shift_DR	4 bits	Remplir $Freg_{cell2}$ Avec la destination Ouest
0	shift_DR			0	shift_DR		
0	shift_DR			0	shift_DR		
1	shift_DR			0	shift_DR		
				0	shift_DR	4 bits	Remplir $Freg_{cell1}$ Avec la destination Nord
				0	shift_DR		
				0	shift_DR		
				0	shift_DR		

Comme c'est montré dans le tableau 2.1, la séquence de bits nécessaire pour remplir le registre Freg de $cell_1$ avec la destination Nord (la première moitié du tableau) a une longueur de (6+5+5+4) bits : 6 bits pour atteindre l'état « Shift IR », 5 bits pour mettre le registre d'instruction de $cell_1$ dans le mode destination, etc. La séquence de bits nécessaire pour remplir le registre Freg de $cell_2$ avec la destination Ouest (deuxième moitié du tableau) a une longueur de (6+5+5+4+4) bits. Ainsi pour configurer ce chemin à deux cellules, le nombre de bits nécessaire est égal à :

$$B_{\text{bits}} = (6+5+5+4) + (6+5+5+4+4) = 44 \text{ bits} \quad (1)$$

En général, pour créer un chemin de N cellules, le nombre de bits à injecter est égal à :

$$B_{\text{bits}} = (6+5+5+4) + (6+5+5+4\times2) + \ldots + (6+5+5+4N) \quad (2)$$

C'est une suite arithmétique de N termes et de raison 4. Le premier terme est égal à (6+5+5+4) =20 et le dernier terme est égal à : (6+5+5+4N) = 16+4N.

Rappel :

Étant donné une suite arithmétique de N termes et de raison r :

$$\sum_{k=1}^{N} U_k = U_1 + U_2 + \cdots + U_N \quad (3)$$

La somme est définie par :

$$\sum_{k=1}^{N} U_k = \frac{N \times (U_1 + U_N)}{2} \quad (4)$$

En appliquant (4) sur (2) et en prenant U_1 = 20, U_N = 16+4N, B_{bits} s'écrit :

$$B_{\text{bits}} = \frac{20+16+4N}{2} \times N \quad (5)$$

Finalement, tout calcul fait on aura :

$$B_{\text{bits}} = 2N^2+18N \quad (6)$$

Pour généraliser on dira que, pour créer un chemin de N cellules il faut envoyer un flux de bits JTAG de longueur $2N^2 + 18N$ bits. L'étude théorique a montré que l'élaboration d'un chemin est d'une complexité $T=O(N^2)$ où N est le nombre de cellules dans le chemin (la longueur du chemin). Le premier objectif du diagnostic consiste à trouver un ensemble de chemins qui couvrent tous les liens et toutes les cellules du réticule en un minimum de temps. Plusieurs couvertures sont envisageables dépendamment de la position des cellules connectées aux ports d'entrée/sortie (TDI/TDO). Dans le prochain paragraphe, deux types de couvertures possibles des liens et des cellules du réticule du WaferIC sont présentées : Couverture par des chemins rectangulaires et couverture par des chemins en serpentins.

2.5 Complexité temporelle de deux couvertures possibles : Couverture par des chemins rectangulaires et couverture par des chemins en serpentins

Pour couvrir les quatre liens sortants de toutes les cellules du réticule, plusieurs formes de chemins peuvent être adoptées. Le but est de trouver un ensemble de chemins qui parcourent tous les éléments du réticule en un minimum de temps. En d'autres termes, le temps de configuration de ces chemins doit être minimal. L'étude réalisée dans ce paragraphe s'applique à un modèle abstrait d'un réticule du WaferIC qui est une matrice de 32×32 cellules ayant les ports TDI-TDO dans le coin inférieur gauche.

Pour parcourir les liens Nord et Sud du réticule, un serpentin vertical (figure 2-5.a) peut être adopté. De même, pour parcourir les liens Est et Ouest un serpentin horizontal (figure 2-5.b) peut être adopté. Dans une matrice de $n \times n$ cellules, 2 serpentins verticaux (figure 2-5.c) sont nécessaires pour parcourir les liens Nord et Sud. Également 2 serpentins horizontaux (figure 2-5.d) sont nécessaires pour parcourir les liens Est et Ouest. Dans ce paragraphe, on se limitera aux serpentins verticaux, mais c'est le même raisonnement qui s'applique pour les serpentins horizontaux.

Un serpentin peut toujours être segmenté en un ensemble de k petits serpentins $\{S_1, S_2, \ldots, S_k\}$. La figure 2-6 montre un exemple de segmentation d'un serpentin dans une grille de 8×8 cellules. Dans la figure 2-6.a le serpentin est segmenté en deux (k=2) tandis que dans la figure 2-6.b le serpentin est segmenté en quatre (k=4). Dans une matrice de 32×32 cellules k peut prendre les valeurs {2, 4, 8 ou 16}.

Cette discussion amène quelques questions quant à savoir s'il est plus bénéfique de configurer un seul serpentin parcourant les liens Nord et Sud du réticule ou s'il faut le segmenter en un ensemble de *k* serpentins $\{S_1, S_2, \ldots, S_k\}$ plus courts. S'il s'avère bénéfique de le segmenter, cela amène la question de savoir quelle serait la valeur de *k* qui correspond à un coût total (nombre de bits) le moins élevé.

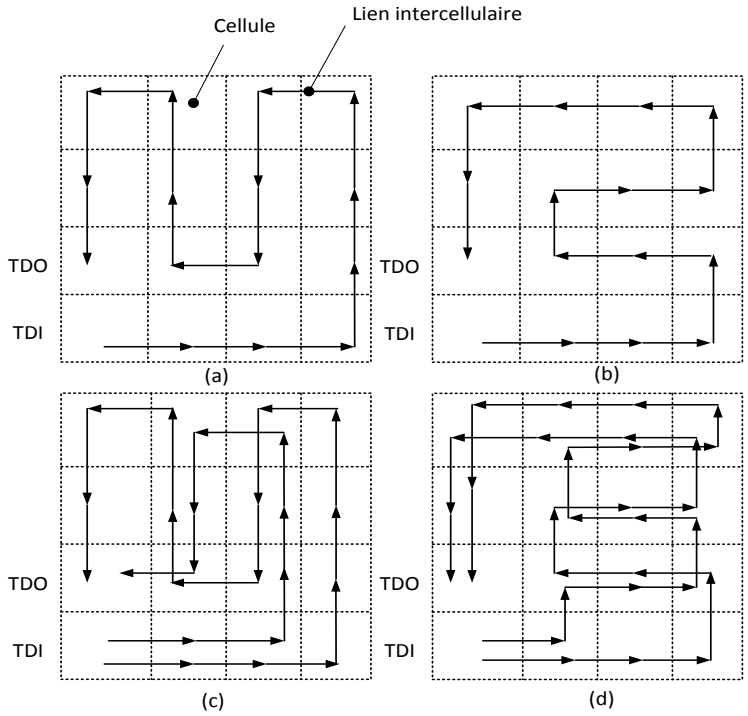


Figure 2-5: Chemins en serpentins (a) serpentin vertical (b) serpentin horizontal (c) deux serpentins verticaux pour parcourir la majorité des liens Nord et Sud (d) deux serpentins horizontaux pour parcourir la majorité des liens Est et Ouest.

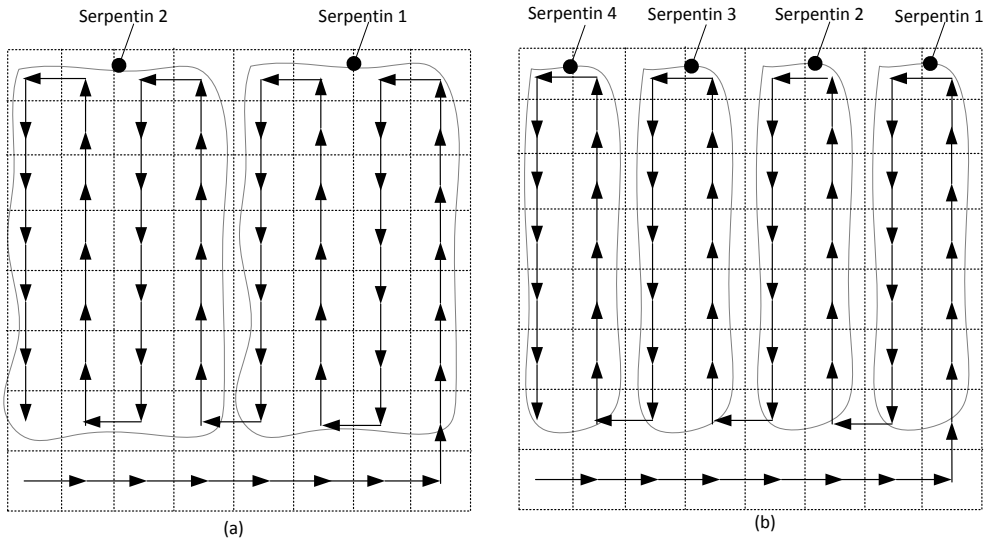


Figure 2-6: Segmentation d'un serpentin (a) Segmentation en deux serpentins (k=2) (b) Segmentation en quatre serpentins (k=4).

Réponse : Soit S le serpentin parcourant la totalité des $N = n^2$ cellules d'une grille de dimensions $n \times n$. D'après l'analyse de complexité établie précédemment, le nombre de bits nécessaires pour configurer S est égal à : $B_{\text{bits (S)}} = 2N^2+18N$. En découpant S en k serpentins $\{S_1, S_2, \ldots, S_k\}$, un serpentin moyen S_i aura une taille plus ou moins égale à $(N/k+\sqrt{N})$.

En effet, comme c'est montré dans la figure 2-7, $\sqrt{N}$ est la taille moyenne du chemin liant la première cellule du serpentin à TDI et sa dernière cellule à TDO et N/k est la taille du serpentin.

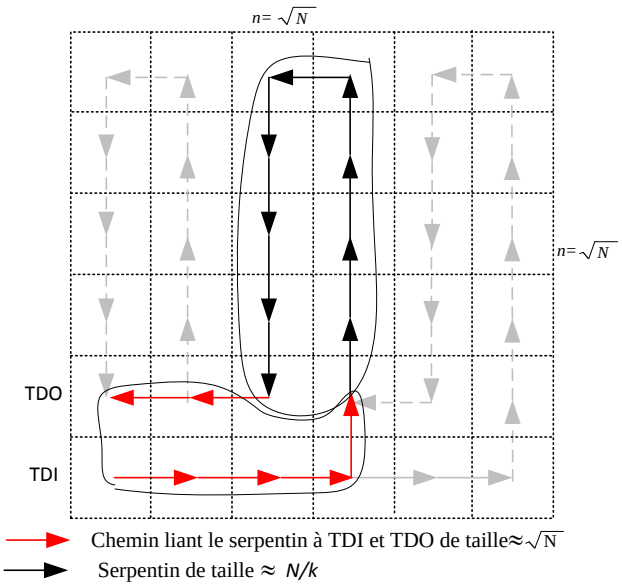


Figure 2-7: Taille moyenne d'un serpentin.

Le nombre de bits nécessaires pour configurer un serpentin S_i de l'ensemble $\{S_1, S_2, \ldots, S_k\}$est plus ou moins égal à : $B_{\text{bits (Si)}} = 2\,(N/k + \sqrt{N})^2 + 18\,(N/k + \sqrt{N})$.

Par conséquent, le nombre de bits nécessaires pour configurer les k serpentins de l'ensemble $\{S_1, S_2, \ldots, S_k\}$ est plus ou moins égal à : $B_{\text{bits (k serpentins)}} = 2k\,(N/k + \sqrt{N})^2 + 18k(N/k + \sqrt{N})$

avec N=1024 et k= {2, 4, 8, 16}.

La figure 2-8 montre le nombre de bits de configuration du serpentin de N cellules ainsi que le nombre de bits de configuration des k serpentins pour des valeurs de k égales à {2, 4, 8, 16}. On remarque à travers ces courbes que configurer plusieurs serpentins est bénéfique en comparaison du cas où on configurerait un seul serpentin passant par la totalité des cellules. On remarque également que diviser le serpentin en k=16 serpentins permet d'avoir le coût de configuration le moins élevé. Diviser le serpentin en 16 permet de passer de 2 115 584 bits à ≈313 344 bits ce qui donne un gain proche de 85,18%. Les serpentins correspondants à k=16 seront appelés rectangles dans le reste du mémoire.

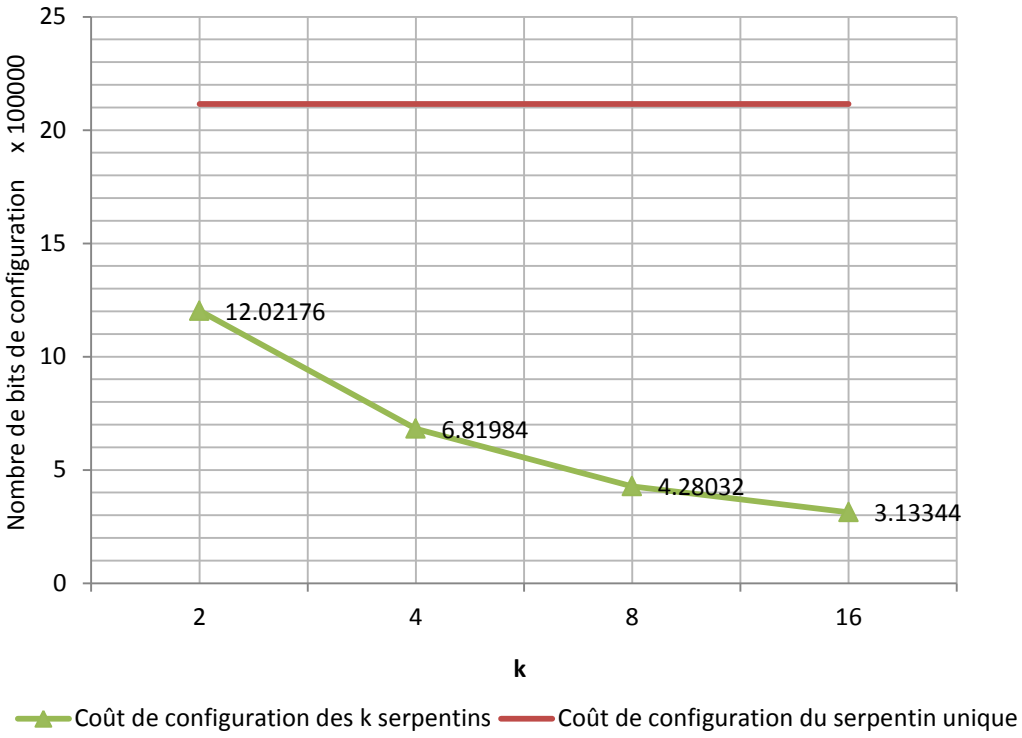


Figure 2-8: Coût de configuration d'un serpentin parcourant toutes les cellules et coût de configuration de *k* serpentins dans un réticule de 1024 cellules.

On remarque qu'un long serpentin passant par toutes les cellules du réticule nécessite un temps de configuration beaucoup plus élevé que celui correspondant aux 16 rectangles. On déduit qu'il est plus bénéfique d'avoir un nombre élevé de chemins relativement courts qu'un nombre très réduit de chemins longs. Cette comparaison a montré que les chemins rectangulaires (correspondant à k=16) permettent de couvrir les liens intercellulaires en un minimum de temps. Ainsi l'algorithme de diagnostic proposé dans ce mémoire utilisera des chemins sous forme de rectangles pour parcourir les liens intercellulaires du réticule. Dans le prochain paragraphe, une étude détaillée sur les chemins rectangulaires est dressée.

2.6 Couverture par des chemins rectangulaires (modélisation avec Matlab)

Pour couvrir tous les liens intercellulaires du réticule, plusieurs chemins peuvent être créés. Ces chemins doivent non seulement balayer toutes les cellules et tous les liens intercellulaires mais aussi en un minimum de temps. Le paragraphe précédent a comparé deux couvertures possibles

des liens : couverture par des chemins en rectangles et couverture par des chemins en serpentins. Il a été démontré que les chemins en rectangle nécessitent un temps de configuration beaucoup plus court. Ce présent paragraphe décrit en détail la couverture par des chemins rectangulaires.

Dans la figure 2-9, une grille de 4×4 cellules est schématisée où une flèche représente un lien intercellulaire liant deux cellules immédiatement voisines. Les rectangles verticaux (Figure 2-9.a) parcourent la majorité des liens Nord et Sud de la grille tandis que les rectangles horizontaux (Figure 2-9.b) parcourent la majorité des liens Est et Ouest de la grille. Chacun de ces rectangles doit être un chemin complet c'est-à-dire commençant par la cellule liée physiquement au port TDI et se terminant par la cellule liée physiquement au port TDO. Dans une grille de $n \times n$ cellules, (n-1) rectangles verticaux et (n-1) rectangles horizontaux sont nécessaires pour parcourir la majorité des liens de la grille. Dans le cas d'un réticule du WaferIC constitué de 32×32 cellules, 31 rectangles verticaux et 31 rectangles horizontaux sont nécessaires pour parcourir la majorité des liens. Dans notre algorithme les chemins rectangulaires ont été créés suivant cet ordre :

1- Création des rectangles verticaux en suivant l'ordre croissant des x.
2- Création des rectangles horizontaux en suivant l'ordre décroissant des y.

Cependant, des chemins supplémentaires doivent être créés pour couvrir 100% des liens. Ceci a été simple à déterminer grâce à la modélisation du réticule et des chemins rectangulaires avec le logiciel MATLAB. Sans la modélisation, il serait également pénible de quantifier le nombre de cellules présentes dans chaque chemin. Connaitre le nombre de cellules dans chaque chemin permet de prédire la longueur des flux de bits JTAG à envoyer et ainsi le temps que va prendre la configuration.

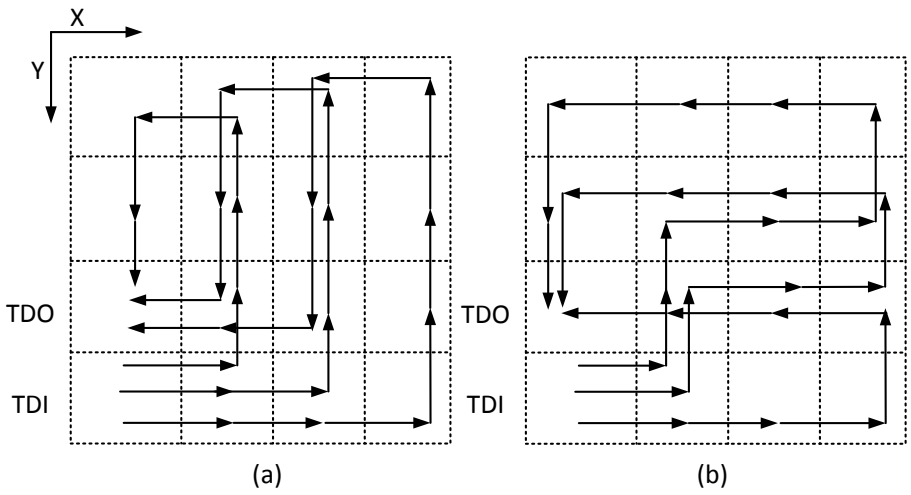


Figure 2-9: Ensemble des chemins en rectangles (a) Rectangles verticaux (b) rectangles horizontaux.

Modélisation des chemins rectangulaires avec Matlab

À l'aide du logiciel Matlab, une matrice de taille 32×32 a été créée. Au total 31 rectangles verticaux et 31 rectangles horizontaux ont été créés. Chaque cellule de la matrice est représentée par un mot de 4 bits $b_1 \ldots b_4$ tel que :

b_1 =1 (donne 8 en décimal) : Si le lien sortant dans la direction Nord de la cellule est parcouru par un rectangle.

b_4 =1 (donne 1 en décimal) : Si le lien sortant dans la direction Est de la cellule est parcouru par un rectangle.

b_2 =1 (donne 4 en décimal) : Si le lien sortant dans la direction Sud de la cellule est parcouru par un rectangle.

b_3 =1 (donne 2 en décimal) : Si le lien sortant dans la direction Ouest de la cellule est parcouru par un rectangle.

Un OU logique est effectué entre toutes les valeurs de la cellule pour déterminer les liens sortants qui ont été parcourus par les chemins rectangulaires. La figure 2-10 montre le résultat donné en

effectuant un OU logique entre toutes les valeurs prises par chaque cellule de la matrice. Le tableau 2.2 explique les résultats de cette modélisation.

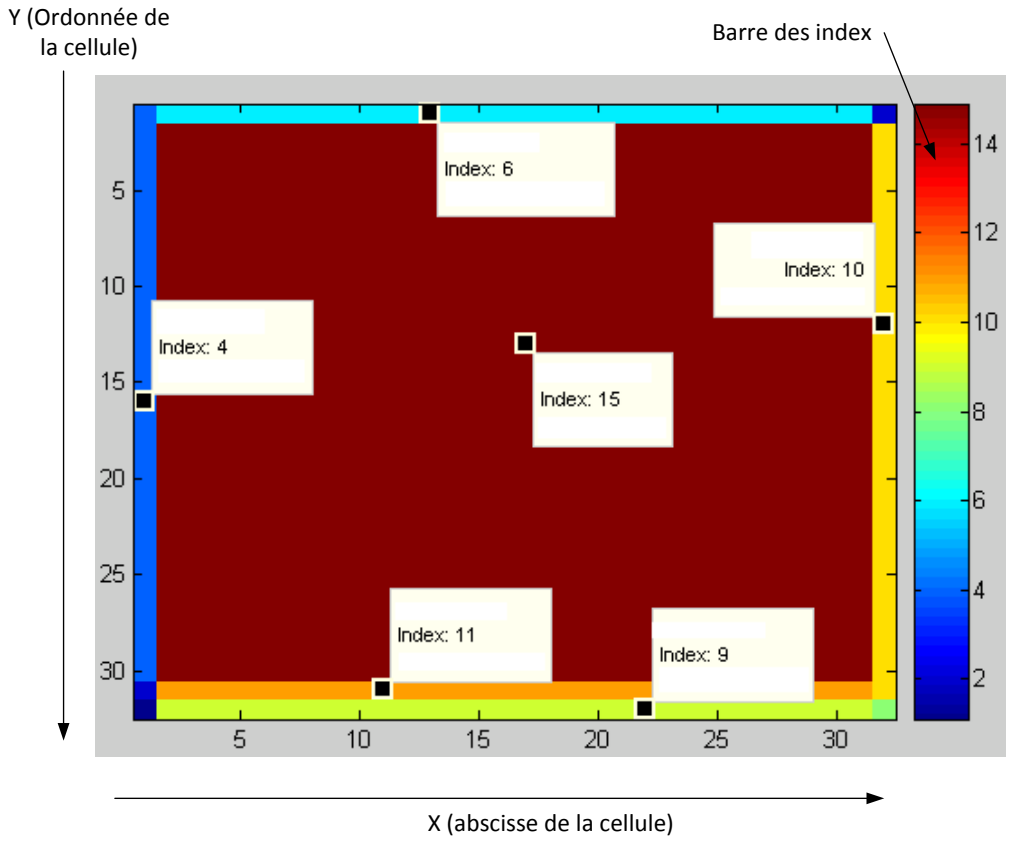


Figure 2-10: Modélisation des 32×32 cellules et des chemins rectangulaires avec le logiciel MATLAB.

Tableau 2.2 Signification des index des cellules.

Index	Signification
15	Tous les liens de la cellule ont été parcourus
4	Les liens SUD de la cellule ont été parcourus (Il manque les liens EST)
6	Les liens OUEST et SUD de la cellule ont été parcourus
10	Les liens NORD et OUEST de la cellule ont été parcourus
9	Les liens NORD et EST ont été parcourus
11	Les liens EST, NORD et OUEST ont été parcourus (Il manque les liens SUD)

Les quatre liens sortants des cellules d'index 15 sont parcourus par des rectangles et par conséquent peuvent être diagnostiqués. Les cellules d'index 4 situées sur le bord gauche de la matrice (X=0), n'ont que le lien Sud qui est parcouru par des rectangles. Les liens dans la direction Est ne sont pas couverts. À rappeler que les liens dans la direction Nord de ces cellules ont été définis comme étant des liens non «diagnosticables» dans le paragraphe 2.2.1. Les cellules situées sur le bord supérieur de la matrice possèdent un index égal à 6. C'est-à-dire que les liens Ouest et Sud sont parcourus par des rectangles. À rappeler que les liens dans la direction Est de ces cellules ont été définis comme étant des liens non «diagnosticables» dans le paragraphe 2.2.1. Les cellules situées sur le bord droit de la matrice ont un index égal à 10. Les liens Nord et Ouest sont parcourus par les rectangles. À rappeler que les liens dans la direction Sud de ces cellules ont été définis comme étant des liens non «diagnosticables». Les cellules situées sur le bord inférieur de la matrice possèdent un index égal à 9. Les liens Nord et Est de ces cellules sont parcourus par les rectangles. À rappeler que les liens dans la direction Ouest de ces cellules ont été définis comme étant des liens non «diagnosticables». Les liens dans les directions Est, Nord et Ouest des cellules ayant un index égal à 11 sont parcourus par les rectangles. Les liens Sud de ces cellules sont des liens «diagnosticables» mais ne sont pas parcourus par aucun des rectangles. On remarque donc que pour avoir une couverture de 100% des liens intercellulaires «diagnosticables» du réticule, en dehors des rectangles il faut aussi créer quelques chemins supplémentaires pour parcourir les liens SUD des cellules d'index 11 (c'est à dire les cellules appartenant à la ligne Y=30) et les liens EST des cellules d'index 4 (c'est à dire les cellules appartenant à la colonne X=0).

Rectangles verticaux

Trente-et-un rectangles verticaux sont nécessaires pour parcourir la majorité des liens Nord et Sud. Le tableau 2.3 montre le nombre des cellules dans chacun des 31 rectangles. Le rectangle vertical le plus court contiendrait 64 cellules tandis que le rectangle vertical le plus long contiendrait 124 cellules. En moyenne, un rectangle vertical contiendrait 94 cellules. Le tableau 2.4 donne le nombre de bits de configuration de chacun des rectangles verticaux.

Tableau 2.3 Nombre de cellules dans chacun des rectangles verticaux.

Chemin	Nombre de cellules	Chemin	Nombre de cellules	Chemin	Nombre de cellules
Chemin 1	64	Chemin 11	84	Chemin 21	104
Chemin 2	66	Chemin 12	86	Chemin 22	106
Chemin 3	68	Chemin 13	88	Chemin 23	108
Chemin 4	70	Chemin 14	90	Chemin 24	110
Chemin 5	72	Chemin 15	92	Chemin 25	112
Chemin 6	74	Chemin 16	94	Chemin 26	114
Chemin 7	76	Chemin 17	96	Chemin 27	116
Chemin 8	78	Chemin 18	98	Chemin 28	118
Chemin 9	80	Chemin 19	100	Chemin 29	120
Chemin 10	82	Chemin 20	102	Chemin 30	122
				Chemin 31	124

Tableau 2.4 Nombre de bits de configuration de chacun des rectangles verticaux.

Chemin	Nombre de bits de configuration	Chemin	Nombre de bits de configuration	Chemin	Nombre de bits de configuration
Chemin 1	9344	Chemin 11	15624	Chemin 21	23504
Chemin 2	9900	Chemin 12	16340	Chemin 22	24380
Chemin 3	10472	Chemin 13	17072	Chemin 23	25272
Chemin 4	11060	Chemin 14	17820	Chemin 24	26180
Chemin 5	11664	Chemin 15	18584	Chemin 25	27104
Chemin 6	12284	Chemin 16	19364	Chemin 26	28044
Chemin 7	12920	Chemin 17	20160	Chemin 27	29000
Chemin 8	13572	Chemin 18	20972	Chemin 28	29972
Chemin 9	14240	Chemin 19	21800	Chemin 29	30960
Chemin 10	14924	Chemin 20	22644	Chemin 30	31964
				Chemin 31	32984

Rectangles horizontaux

Trente-et-un rectangles horizontaux sont nécessaires pour parcourir la majorité des liens Est et Ouest. Le tableau 2.5 montre le nombre des cellules dans chacun des 31 rectangles. Le rectangle

horizontal le plus court contiendrait 64 cellules tandis que le rectangle horizontal le plus long contiendrait 124 cellules. En moyenne, un rectangle horizontal contiendrait 94 cellules. Le tableau 2.6 donne le nombre de bits de configuration de chacun des rectangles horizontaux.

Tableau 2.5 Nombre de cellules dans chacun des rectangles horizontaux.

Chemin	**Nombre de cellules**	**Chemin**	**Nombre de cellules**	**Chemin**	**Nombre de cellules**
Chemin 1	64	Chemin 11	84	Chemin 21	104
Chemin 2	66	Chemin 12	86	Chemin 22	106
Chemin 3	68	Chemin 13	88	Chemin 23	108
Chemin 4	70	Chemin 14	90	Chemin 24	110
Chemin 5	72	Chemin 15	92	Chemin 25	112
Chemin 6	74	Chemin 16	94	Chemin 26	114
Chemin 7	76	Chemin 17	96	Chemin 27	116
Chemin 8	78	Chemin 18	98	Chemin 28	118
Chemin 9	80	Chemin 19	100	Chemin 29	120
Chemin 10	82	Chemin 20	102	Chemin 30	122
				Chemin 31	124

Tableau 2.6 Nombre de bits de configuration de chacun des rectangles horizontaux.

Chemin	**Nombre de bits de configuration**	**Chemin**	**Nombre de bits de configuration**	**Chemin**	**Nombre de bits de configuration**
Chemin 1	9344	Chemin 11	15624	Chemin 21	23504
Chemin 2	9900	Chemin 12	16340	Chemin 22	24380
Chemin 3	10472	Chemin 13	17072	Chemin 23	25272
Chemin 4	11060	Chemin 14	17820	Chemin 24	26180
Chemin 5	11664	Chemin 15	18584	Chemin 25	27104
Chemin 6	12284	Chemin 16	19364	Chemin 26	28044
Chemin 7	12920	Chemin 17	20160	Chemin 27	29000
Chemin 8	13572	Chemin 18	20972	Chemin 28	29972
Chemin 9	14240	Chemin 19	21800	Chemin 29	30960
Chemin 10	14924	Chemin 20	22644	Chemin 30	31964
				Chemin 31	32984

Pour un réticule de 32×32 cellules, 31 rectangles verticaux et 31 rectangles horizontaux sont nécessaires pour avoir une couverture totale des cellules. En général pour un réticule de $n \times n$ cellules, (n-1) rectangles verticaux et (n-1) rectangles horizontaux sont nécessaires pour avoir une couverture totale des cellules. En moyenne un rectangle (vertical ou horizontal) contiendrait 94 cellules. En utilisant l'équation (6) établie lors de l'analyse de complexité, en moyenne un flux de données de 19 364 bits est nécessaire pour créer un rectangle. Au total, il faudrait 1 240 248 bits pour configurer les 62 rectangles.

2.7 Sommaire

Au cours de ce chapitre, les objectifs de l'algorithme de diagnostic ont été présentés. L'objectif principal étant de trouver le plus d'éléments (cellules/liens) fonctionnels dans un réticule. Le second objectif étant de minimiser le temps de diagnostic. Comme première étape, il faut trouver un ensemble de chemins qui couvrent toutes les directions de chacune des cellules en un minimum de temps. Il a été démontré que la complexité de création d'un chemin de N cellules est de $O(N^2)$. Suite à l'analyse de complexité élaborée, deux types de couvertures ont été comparées : La couverture par des chemins rectangulaires et la couverture par des chemins en serpentins. Les chemins en rectangles ont été adoptés dans notre algorithme de diagnostic car ils couvrent presque tous les liens en un minimum de temps. La prochaine étape de l'algorithme consiste à envoyer des données à travers les rectangles crées et d'observer la sortie de chacun: si elle correspond à ce qui est attendu alors tous les éléments à savoir les cellules et les liens intercellulaires constituants ce rectangle sont caractérisés comme étant fonctionnels. Dans le cas où la sortie ne correspond pas à ce qui est attendu, la conclusion qu'on peut tirer est qu'il existe un ou plusieurs éléments défectueux dans ce rectangle non fonctionnel. Tous les éléments du chemin non fonctionnel sont susceptibles d'être défectueux et par conséquent ne sont pas des ressources utilisables. Pour augmenter l'espace des éléments fonctionnels qui peuvent être utilisés pour faire les interconnexions nécessaires au sein du WaferIC, un algorithme de recherche, basé sur la dichotomie, est appliqué sur chaque chemin non fonctionnel pour localiser le(s) lien(s) défectueux. Le chapitre 3 décrit en détail l'algorithme de diagnostic par dichotomie. Il présente également des algorithmes heuristiques, complémentaires à l'algorithme de dichotomie, qui ont pour but d'améliorer la résolution du diagnostic en localisant le plus précisément possible le(s) lien(s) suspect(s) d'être défectueux.

CHAPITRE 3 MÉTHODE ALGORITHMIQUE POUR LE DIAGNOSTIC DE LA CHAÎNE JTAG

3.1 Introduction

Des données sont envoyées à travers chaque chemin rectangulaire crée et la sortie au niveau du port TDO est observée. Si les données envoyées apparaissent au niveau du TDO alors le chemin a conduit le signal depuis sa cellule d'entrée TDI jusqu'à sa cellule de sortie TDO et par conséquent toutes les cellules et tous les liens du chemin sont fonctionnels. Dans le cas contraire, si les données sont erronées ou bien n'apparaissent pas au niveau du port TDO alors le chemin contient un ou plusieurs éléments (cellules ou liens) défectueux. Dans ce cas le chemin sera analysé pour localiser les éléments fonctionnels et les éléments défectueux (ou probablement défectueux). Cette analyse est faite par un algorithme de recherche basé sur le principe de dichotomie. Cet algorithme tente de localiser les liens fonctionnels ainsi que les liens défectueux ou probablement défectueux au sein d'un chemin non fonctionnel. L'état des cellules sera déduit à partir des liens. Deux algorithmes heuristiques complémentaires à l'algorithme de dichotomie permettent d'améliorer la résolution du diagnostic et cibler la recherche de l'élément défectueux.

Ce chapitre se divise en quatre parties qui :

- Présentent une technique qui pourrait être utilisée pour distinguer les liens fonctionnels des liens défectueux au sein d'un chemin non fonctionnel. Cette technique se base sur la théorie des ensembles (Intersection, union des chemins fonctionnels et non fonctionnels).
- Décrivent l'organigramme de l'algorithme de dichotomie.
- Décrivent les algorithmes HEURISTIC_1 et HEURISTIC_2.
- Décrivent l'algorithme basé sur le routage de LEE.

3.2 Recherche des éléments défectueux en se basant sur la théorie des ensembles

Il serait impossible de localiser le(s) élément(s) défectueux au sein d'un chemin non fonctionnel sans segmenter le chemin en question ou bien créer un ensemble de plusieurs chemins qui

passent à travers ses différents éléments. Dans ce cas, l'objectif est de minimiser au maximum la taille totale des flux de bits JTAG requis pour configurer cet ensemble de chemins.

Parmi les techniques qui peuvent être utilisées pour localiser les liens intercellulaires avec une forte probabilité d'être défectueux, il y a la méthode basée sur la théorie des ensembles (intersection, union, exclusion des chemins fonctionnels et non fonctionnels). L'exemple de la figure 3-1 illustre cette technique. Soit le chemin (Ch 1) de la figure 3-1. Ce chemin commence par la cellule TDI, parcourt un certain nombre de cellules jusqu'à arriver à la cellule TDO. Le chemin (Ch 1) contient 8 cellules, 7 liens intercellulaires L_1= {l_1, l_2, l_3, l_4, l_5, l_6 et l_7} et deux liens JTAG {l_0 et l_8}. Une fois ce chemin est planifié, des données sont envoyées à travers ses cellules et ses liens intercellulaires. Supposons que ces données ne ressortent pas par TDO, donc le chemin est non fonctionnel et il y a 3 possibilités :

- Soit il y a un ou plusieurs liens intercellulaires défectueux dans le chemin.
- Soit il y a une ou plusieurs cellules défectueuses dans le chemin.
- Soit l'un ou les deux liens JTAG sont défectueux.

Pour l'instant aucune conclusion sur les liens intercellulaires ne peut être tirée.

Un second chemin (Ch 2) est planifié. Ce chemin contient 8 cellules et 7 liens intercellulaires dont 5 font aussi partie du chemin (Ch 1). Des données sont envoyées à travers ce chemin. Supposons que ces données arrivent à sortir de TDO, donc le chemin (Ch 2) est fonctionnel. Ceci signifie que toutes les cellules et tous les liens faisant partie du chemin (Ch 2) sont fonctionnels. Et par conséquent les liens {l_1, l_2, l_3, l_4 et l_5} sont fonctionnels. Ainsi la défectuosité se trouve dans l'ensemble {Ch1 $\not\subset$ Ch 2}. Par conséquent, le lien l_6 ou bien le lien l_7 est défectueux. Pour savoir lequel des liens {l_6, l_7} est défectueux, un troisième chemin (Ch 3) est alors planifié. Il contient 6 cellules et 5 liens intercellulaires dont le lien l_7. Supposons que ce chemin est fonctionnel, c'est-à-dire que toutes ses cellules et tous ses liens dont l_7 sont non bloquants. La conclusion qu'on peut tirer de ces trois chemins, est que le lien défectueux est le lien l_6. Ainsi moyennant 3 chemins distincts on n'a pu localiser le lien défectueux. L'exemple de la figure 3-1 a montré qu'il a fallu créer deux chemins supplémentaires (Ch 2) et (Ch 3) pour localiser la défectuosité se trouvant dans le chemin non fonctionnel (Ch 1). Dans le cas où le nombre de défectuosités dans le circuit est faible, la méthode de la théorie des ensembles est efficace. Dans

le cas où le circuit contient un nombre élevé de liens défectueux, cette méthode demande la configuration d'un grand nombre de chemins. Il faut aussi tenir compte du fait que la taille totale des flux de bits JTAG requis pour configurer ces chemins doit être minimale.

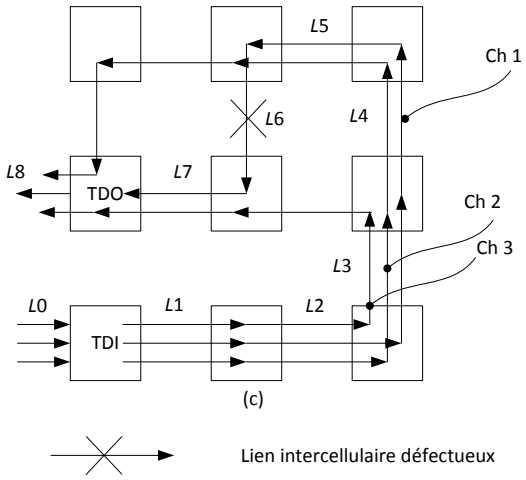


Figure 3-1: Application de la théorie des ensembles pour identifier un lien intercellulaire défectueux.

3.3 L'algorithme de diagnostic par dichotomie

Un algorithme de recherche basé sur la dichotomie a été implémenté pour localiser le(s) lien(s) défectueux se trouvant sur un chemin non fonctionnel. Pour améliorer la résolution du diagnostic, deux heuristiques ont également été implémentées pour compléter l'algorithme de dichotomie.

Afin de repérer les éléments (cellules/liens) défectueux, l'algorithme de dichotomie cherche la défectuosité au niveau des liens intercellulaires. En effet, l'état des cellules sera déduit à partir des liens : Si une cellule possède quatre liens intercellulaires entrants défectueux ou quatre liens intercellulaires sortants défectueux alors la cellule est nécessairement défectueuse.

La figure 3-11 (page 72) montre l'organigramme de cet algorithme. C'est un algorithme récursif qui reçoit en paramètre le chemin non fonctionnel (NFP) et retourne le lien défectueux. Le chemin non fonctionnel (NFP) est divisé en deux segments s_1 et s_2.

Division d'un chemin en deux segments

Intuitivement, le principe de dichotomie amène à diviser le chemin en deux segments de longueurs égales. Cependant les deux segments du chemin ne doivent pas être tout à fait égaux. En effet, le deuxième segment doit contenir la dernière cellule du premier segment pour ne pas perdre le lien entre les deux. La figure 3-2 est un exemple qui illustre la segmentation du chemin. Étant donné le chemin constitué de six cellules et cinq liens intercellulaires montré dans la figure 3-2.a. Intuitivement, diviser ce chemin en deux, amène à deux segments s_1 et s_2 chacun constitué de 3 cellules (figure 3-2.a). Or pour tester chacun de ces segments, il faut le lier aux cellules TDI-TDO par des chemins tels que représentés dans la figure 3-2.b. On remarque que le lien l_3 n'appartient ni au segment s_1 ni au segment s_2. Ce lien sera donc non testé. Pour pallier ce problème, le segment 2 doit contenir la dernière cellule du segment 1 comme c'est indiqué dans la figure 3-2.c. Le segment 2 contiendra donc 4 cellules et 3 liens tandis que le segment 1 contient 3 cellules et 2 liens.

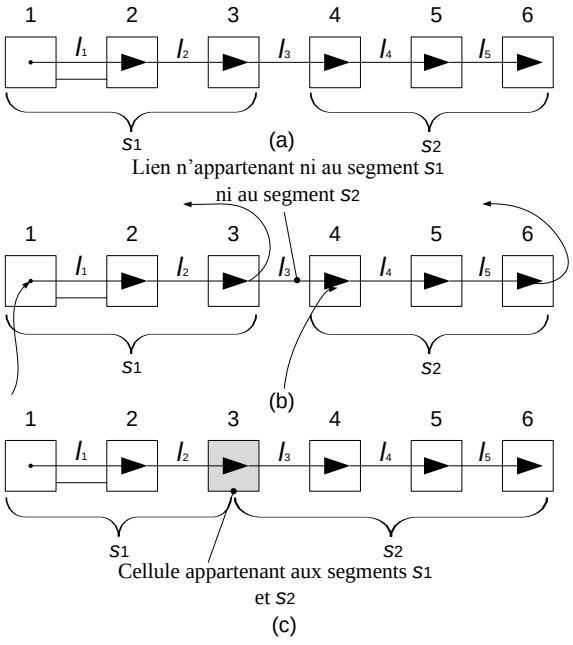


Figure 3-2: Division d'un chemin en deux segments.

Cependant, dans le reste de ce mémoire, toutes les figures explicatives sont schématisées avec s_1 et s_2 de tailles égales afin de faciliter la compréhension.

Deux fonctions principales apparaissent dans l'organigramme de l'algorithme de dichotomie donné à la figure 3-11 de la page 72.

La fonction CONNECT : est basée sur l'algorithme de routage de LEE. Elle retourne un chemin complet nommé p_i qui commence par la cellule TDI couvre le segment s_i et se termine par la cellule TDO. En effet, pour tester un segment s_i un chemin liant la cellule TDI à sa première cellule ainsi qu'un chemin liant sa dernière cellule à la cellule TDO (Figure 3-3) doivent être crées pour fermer le chemin et tester s_i. Les liens utilisés pour connecter s_i à TDI et TDO doivent être préalablement caractérisés comme fonctionnels et leur nombre doit être le plus petit possible afin de minimiser la taille du flux de bits JTAG.

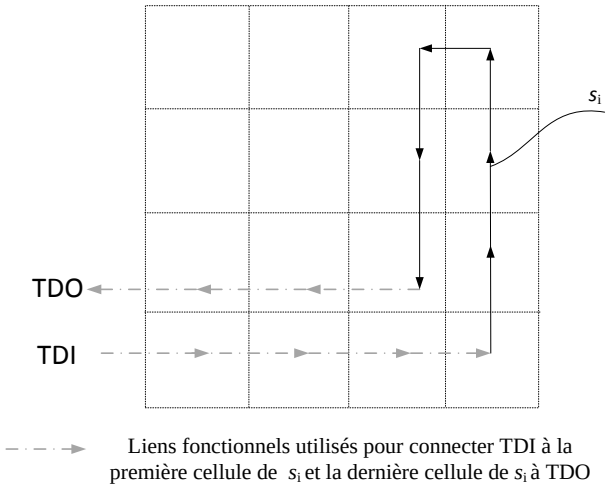


Figure 3-3: Connexion d'un segment aux cellules TDI et TDO.

Si la fonction CONNECT n'arrive pas à lier un segment donné à TDI et TDO à cause d'éléments défectueux ou potentiellement défectueux bloquants, alors cette fonction ne retourne aucun chemin (la taille du chemin p_i est égale à zéro). Quelques autres algorithmes pourraient être adoptés pour trouver un chemin fonctionnel entre deux points donnés. Parmi ces algorithmes, il y a d'autres algorithmes de routage en labyrinthes (*maze routing algorithms*) et des algorithmes de routage en lignes (*line probe algorithms*). Le choix de l'algorithme de LEE revient essentiellement à :

- Parmi les algorithmes de routage en labyrinthes, c'est le plus simple à implémenter.

- C'est une version améliorée de l'algorithme de parcours en largeur (*Breadth-first search*) donc il garantit de trouver un chemin fonctionnel entre deux points si un tel chemin existe. Le chemin trouvé sera le chemin le plus court.

L'implémentation de l'algorithme de LEE est donnée en détail plus loin dans ce chapitre.

La fonction TEST : Reçoit en paramètre un chemin complet et retourne vrai si ce chemin délivre un flux de bits non corrompu (chemin fonctionnel). Dans le cas contraire (chemin non fonctionnel), cette fonction retourne faux.

Une fois le chemin non fonctionnel (NFP) est divisé en deux segments s_1 et s_2, chacun des segments est envoyé à la fonction CONNECT pour être lié, par des liens fonctionnels, à TDI-TDO pour avoir des chemins complets p_1 et p_2. Dans notre algorithme p_2 est testé en premier. Trois cas de figures se présentent :

1er cas : s_2 a été connecté avec succès à TDI et TDO et p_2 est fonctionnel alors tous les éléments de s_2 sont caractérisés comme fonctionnels et s_1 sera segmenté.

2éme cas : s_2 a été connecté avec succès à TDI et TDO et p_2 est non fonctionnel alors s_2 sera segmenté. Pour collecter le maximum d'informations sur les liens fonctionnels et les liens défectueux (ou probablement défectueux), le segment s_1 doit aussi être testé. Ici trois sous cas concernant s_1 se présentent :

- 1er sous cas : s_1 a été connecté avec succès à TDI-TDO et p_1 est fonctionnel alors tous les éléments de s_1 sont caractérisés comme fonctionnels.
- 2éme sous cas : s_1 a été connecté avec succès à TDI-TDO et p_1 est non fonctionnel alors s_1 est inséré dans une liste nommée *nonfunctional_s_list* pour être analysé plus tard par l'algorithme de dichotomie. Ceci aide à localiser tous les liens défectueux qui peuvent se trouver dans un même chemin.
- 3éme sous cas : s_1 n'a pas été connecté avec succès à TDI-TDO alors s_1 est sauvegardé dans une liste nommée *unconnected_s_list.* Les segments de cette liste seront connectés aux cellules TDI-TDO et testés à la fin de l'algorithme quand plus d'éléments fonctionnels auront été trouvés.

3éme cas : s_2 ne peut pas être connecté à TDI-TDO alors trois sous cas se présentent :

- 1er sous cas : s_1 a été connecté avec succès à TDI-TDO et p_1 est fonctionnel alors tous les éléments de s_1 sont caractérisés comme fonctionnels et s_2 sera segmenté.
- 2éme sous cas : s_1 a été connecté avec succès à TDI-TDO et p_1 est non fonctionnel alors s_1 sera segmenté et s_2 sera ajouté à la liste *unconnected_s_list*.
- 3éme sous cas : s_1 n'a pas été connecté avec succès à TDI-TDO alors l'algorithme de dichotomie s'arrête et l'ensemble $\{s_1, s_2\}$ sera sauvegardé dans une liste nommée *heuristic_list* pour être traité par un algorithme heuristique (décrit dans le paragraphe suivant).

3.4 Diagnostic par des algorithmes heuristiques

3.4.1 Premier algorithme heuristique : HEURISTIC_1

À défaut de liens fonctionnels disponibles, il arrive que la fonction CONNECT de l'algorithme de dichotomie ne soit pas capable de connecter ni le segment s_1 ni le segment s_2 d'un chemin non fonctionnel (NFP) aux cellules TDI-TDO. Par conséquent les deux segments ne peuvent pas être testés puisqu'aucun flux de données ne peut être envoyé à travers eux. Un exemple de ce cas de figure est donné en annexe A. À ce stade, il n'est plus possible de repérer lequel des deux segments est non fonctionnel pour pouvoir le découper. Les liens des segments s_i qui n'ont pas été caractérisés comme fonctionnels par d'autres chemins, demeurent dans un état inconnu. On ne peut pas affirmer s'ils sont fonctionnels ou défectueux. On dira que tous les liens intercellulaires appartenant aux deux segments s_1 et s_2 et qui n'ont pas été caractérisés comme fonctionnels par d'autres chemins, sont potentiellement défectueux. Pour réduire le nombre de liens potentiellement défectueux, un algorithme heuristique a été implémenté nommé HEURISTIC_1 et il est appliqué sur l'ensemble $\{s_1, s_2\}$. Il prend individuellement chaque lien de l'ensemble et le connecte à TDI-TDO par des éléments fonctionnels (en utilisant la fonction CONNECT) et teste le chemin résultant. Si ce chemin est fonctionnel alors le lien qu'il couvre est fonctionnel sinon le lien est défectueux. Dans le cas où il n'y aucun chemin complet possible couvrant un lien donné, alors ce lien demeure potentiellement défectueux.

La figure 3-4 montre un exemple de cet algorithme heuristique. Soit le chemin non fonctionnel (NFP) constitué des quatre liens intercellulaires potentiellement défectueux $\{l_1, l_2, l_3$ et $l_4\}$. L'algorithme HEURISTIC_1 prend chaque lien et le complète par des liens fonctionnels afin

d'avoir des chemins complets {ch_1, ch_2, ch_3 et ch_4}. Chacun des chemins ch_i sera par la suite testé. Si ch_i est fonctionnel alors le lien l_i qu'il couvre est fonctionnel sinon le lien l_i est défectueux. Si aucun chemin complet possible couvrant un lien l_i ne peut être trouvé alors ce lien demeure dans l'état potentiellement défectueux. Supposons que les chemins {ch_1, ch_2 et ch_3} ont été trouvés fonctionnels alors les liens intercellulaires {l_1, l_2 et l_3} passent de l'état « potentiellement défectueux » à « fonctionnels ». Et supposons que le chemin ch_4 a été trouvé non fonctionnel alors le lien intercellulaire l_4 passe de l'état « potentiellement défectueux » à « défectueux ». La figure 3-5 décrit le pseudo code de l'algorithme HEURISTIC_1 et la figure 3-6 décrit le pseudo code du diagnostic utilisant l'algorithme de dichotomie et l'algorithme HEURISTIC_1.

L'algorithme HEURISTIC_1, appliqué suite à l'algorithme de dichotomie, permet soit de localiser exactement le(s) lien(s) défectueux recherché(s) ou bien discerner le plus étroitement possible la zone contenant le(s) lien(s) recherché(s) en diminuant le nombre de liens potentiellement défectueux.

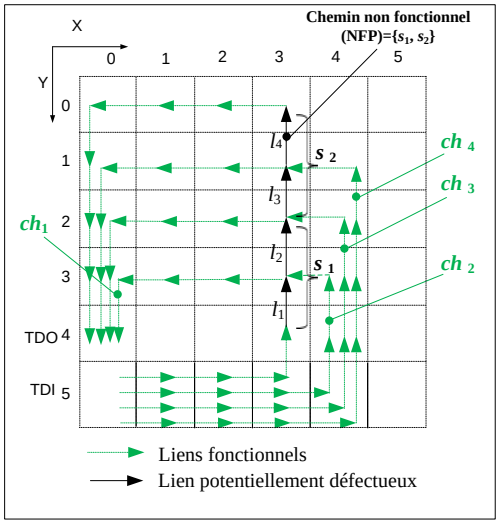


Figure 3-4: Exemple d'application de l'algorithme HEURISTIC_1.

```
1:   Début
2:   Pour chaque l_i de NFP faire
3:   Si ((Statut (l_i) !=Fonctionnel)&(Statut (l_i) !=défectueux))
4:     chemin _i=CONNECT(l_i,TDI,TDO);
5:       Si (chemin _i.taille!=0) & (TEST(chemin _i)=vrai)
6:          Caractériser l_i comme fonctionnel;
7:       sinon si (chemin _i.taille!=0) & (TEST (chemin _i)=faux)
8:          Caractériser l_i comme défectueux;
9:       sinon l_i est potentiellement défectueux;
10:      Fin Si
11:   Fin Si
12:  Fin Pour
13:  Fin
```

Figure 3-5: Pseudo code de l'algorithme HEURISTIC_1.

```
1:   Début
2 :  Configurer les rectangles;
3 :  Pour chaque Rectangle _i faire
4 :    Si (TEST (Rectangle _i) = vrai)
5 :    Caractériser les éléments de Rectangle _i comme fonctionnels;
6 :    Sinon
7 :    Ajouter Rectangle _i à nonfunctional_Rect_list;
8 :    Fin Si
9 :  Fin Pour
10:    Pour chaque Rectangle _i de nonfunctional_Rect_list faire
11:     DICHOTOMIQUE (Rectangle _i);
12:    Fin Pour
13:    Si (nonfunctional_s_list.taille >0)
14:        Pour chaque chemin _i de nonfunctional_s_list faire
15:            DICHOTOMIQUE (chemin _i);
16:        Fin Pour
17:    Fin Si
18:    Si (heuristic_list.taille >0)
19:         HEURISTIC_1 (heuristic_list);
20:    Fin Si
21:    Si (unconnected_s_list.taille >0)
22:        Pour chaque chemin _i de unconnected_s_list faire
23:        p_i=CONNECT (chemin _i, TDI,TDO);
24:            Si (p_i est complet et TEST (p_i)=vrai) alors
25:            Caractériser les éléments de chemin _i comme fonctionnels;
26:            Sinon Si (p_i est complet et TEST (p_i)=faux) alors
27:            DICHOTOMIQUE (chemin _i);
28:            Sinon
29:            Insérer ((chemin _i), list_0);
30:            HEURISTIC_1 (list_0) ;
31:            Fin Si
32:        Fin Pour
33:    Fin Si
34:  Fin
```

Figure 3-6: Pseudo code de l'algorithme de diagnostic.

Comme c'est mentionné plus haut, lorsque le segment s_2 est trouvé non fonctionnel il sera découpé par l'algorithme de dichotomie. Cependant le segment s_1 doit aussi être testé. S'il est trouvé non fonctionnel alors il sera sauvegardé dans la liste *nonfunctional_s_list* pour être analysé plus tard par l'algorithme de dichotomie. Comme c'est montré dans la figure 3-6, les segments s_1 présents dans cette liste seront traités par l'algorithme de dichotomie avant d'appliquer éventuellement l'algorithme heuristique sur les segments de la liste *heuristic_list*. En effet, après l'appel à l'algorithme de dichotomie, l'espace des éléments fonctionnels se voit élargi. Par conséquent il y aura plus de liens fonctionnels disponibles qui peuvent être utilisés par l'algorithme heuristique pour connecter chaque lien potentiellement défectueux à TDI-TDO afin de créer des chemins complets testables (à travers lesquels des données peuvent être envoyées).

Les chemins présents dans la liste *unconnected_s_list* représentent les segments qui n'ont pas pu être connectés à TDI-TDO sans causer l'arrêt de l'algorithme de dichotomie car l'autre segment a été testé et trouvé non fonctionnel. Les segments présents dans cette liste sont traités à la fin de l'algorithme de diagnostic. En effet, après l'appel à l'algorithme de dichotomie et éventuellement à l'algorithme heuristique, le nombre de liens fonctionnels se voit augmenté et par conséquent les chemins qui n'ont pas pu être connectés à TDI-TDO au début de l'algorithme de diagnostic peuvent être connectés avec succès à la fin de l'algorithme. Chaque *chemin* $_{\text{i}}$ de la liste *unconnected_s_list* est lié à TDI-TDO par des liens fonctionnels pour avoir un chemin complet p_{i} qui sera testé. Si p_{i} est fonctionnel alors tous les éléments de *chemin* $_{\text{i}}$ seront caractérisés comme fonctionnels sinon si p_{i} est non fonctionnel alors *chemin* $_{\text{i}}$ sera analysé par l'algorithme de dichotomie afin de localiser le(s) élément(s) défectueux. Si aucun chemin complet possible p_{i} ne peut être trouvé alors l'algorithme HEURISTIC_1 sera appliqué sur *chemin* $_{\text{i}}$ en testant ses liens un par un.

Dans le cas spécifique où le réticule contiendrait un seul lien défectueux, et si ce lien n'est pas précisément localisé par l'algorithme de dichotomie, l'application de l'algorithme HEURISTIC_1 identifie un grand nombre de liens potentiellement défectueux. Pour illustrer ceci, il vaut mieux partir d'un exemple soit celui de la figure 3-7. Le lien défectueux Sud sortant de la cellule (0,1) rend les chemins rectangulaires (R1, R2 et R3) non fonctionnels (figure 3-7 étape 1). On applique en premier lieu l'algorithme de dichotomie sur le rectangle R1 pour tenter de localiser son lien défectueux. Le rectangle R1 est alors divisé en deux segments comme le

montre l'étape 2 de la figure. Le segment 1 ne peut pas être lié à TDO par des liens fonctionnels puisque les liens (1,1) Ouest, (1,1) Nord et (1,1) Est sont potentiellement défectueux car ils appartiennent respectivement au rectangle non fonctionnel R2, au segment 2 et au rectangle non fonctionnel R3. Pareil pour le segment 2, il ne peut pas être lié à TDI par des liens fonctionnels car le lien Nord sortant de la cellule (1,2) est potentiellement défectueux (appartient au segment 1) et le lien sortant Ouest de la cellule (2,1) est aussi potentiellement défectueux (appartient au rectangle R2). Puisqu' aucun des deux segments ne peut être lié à TDI-TDO alors l'ensemble {segment 1, segment 2} est ajouté à la liste *heuristic_list*. Pareil pour les rectangles R2 et R3. Chacun est divisé par l'algorithme de dichotomie jusqu'à ce que les deux segments ne puissent pas être complétés par TDI-TDO (étapes 3 et 4), autrement dit jusqu'à ce que l'algorithme de dichotomie arrive à sa limite. L'ensemble des deux segments est ajouté à la liste *heuristic_list*. Tous les liens contenus dans la liste *heuristic_list* (montrés dans les étapes 2, 3 et 4) et qui n'appartiennent pas à d'autres chemins rectangulaires fonctionnels sont donc potentiellement défectueux. Comme mentionné dans le paragraphe précèdent, une fois que tous les rectangles non fonctionnels sont analysés par l'algorithme de dichotomie, l'algorithme HEURISTIC_1 est appliqué sur chaque segment contenu dans *heuristic_list* afin de réduire le nombre de liens potentiellement défectueux et améliorer la résolution du diagnostic. Tout d'abord, les liens du rectangle R1 montrés à l'étape 2 sont analysés par l'algorithme HEURISTIC_1. Il prend chaque lien qui n'a pas été caractérisé comme fonctionnel et le relie à TDI-TDO (en utilisant des liens fonctionnels) puis teste le chemin résultant. Il y a cinq liens (montrés dans l'étape 5) dont il est impossible de trouver des chemins fonctionnels les liants à TDI-TDO. Ils restent donc dans l'état « potentiellement défectueux ».

L'algorithme HEURISTIC_1 est appliqué ensuite sur les liens montrés à l'étape 3. Chaque lien non caractérisé comme fonctionnel est complété par TDI-TDO afin d'être testé. Deux liens (montrés dans l'étape 6) ne peuvent pas être liés à TDI_TDO. Ils restent donc dans l'état « potentiellement défectueux ».

Finalement, l'algorithme HEURISTIC_1 est appliqué sur les liens montrés à l'étape 4. Chaque lien non caractérisé comme fonctionnel est complété par TDI-TDO afin d'être testé. Trois liens (montrés dans l'étape 7) ne peuvent pas être liés à TDI_TDO. Ils restent donc dans l'état « potentiellement défectueux ».

Au total il y a six liens potentiellement défectueux localisés après l'application de l'algorithme de dichotomie et l'algorithme HEURISTIC_1. Si l'ordonnée (y) de la cellule ayant le lien sortant sud défectueux augmente, on aura un nombre plus élevé de liens « potentiellement défectueux » (car il y aura plus de rectangles horizontaux non fonctionnels). Une défectuosité au niveau du lien intercellulaire Sud sortant de la cellule (0, y) rend y+2 rectangles non fonctionnels.

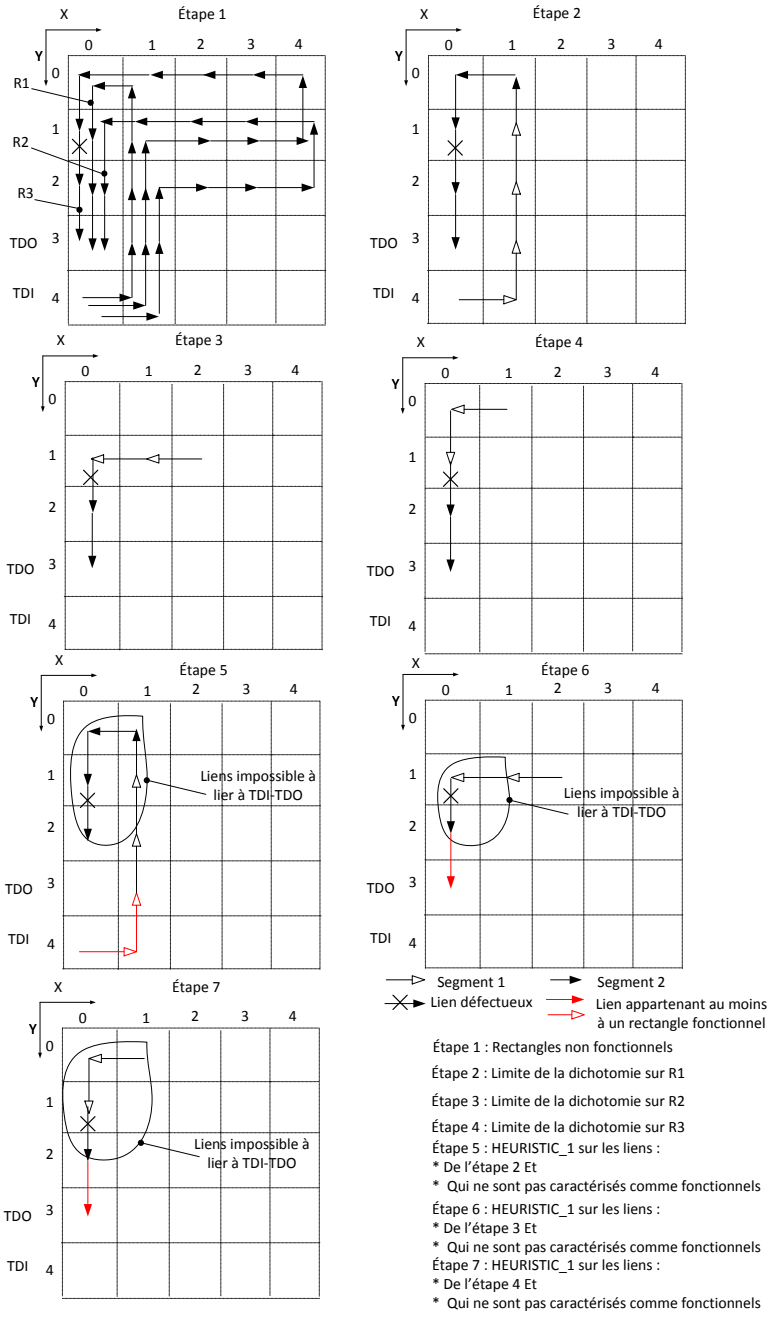


Figure 3-7: Localisation d'un lien défectueux par dichotomie et HEURISTIC_1.

L'application de l'algorithme HEURISTIC_1 après celui de la dichotomie, fait baisser le nombre de liens potentiellement défectueux à 6. Le lien défectueux figure donc parmi ces 6 liens. Afin de cibler la recherche encore plus et rétrécir le nombre de liens parmi lesquels figure la défectuosité, il est préférable d'appliquer, au lieu de HEURISTIC_1, un autre algorithme heuristique après l'algorithme de dichotomie. Cet algorithme est basé sur le principe d'intersection et est nommé HEURISTIC_2.

3.4.2 Deuxième algorithme heuristique : HEURISTIC_2 (cas d'un seul lien défectueux)

L'algorithme HEURISTIC_2 est basé sur le principe de l'intersection et est appelé dans le cas où :

- Un seul lien défectueux est présent dans le réticule.
- Après l'analyse des rectangles non fonctionnels par l'algorithme de dichotomie (Figure 3-6 ligne 12), celui-ci identifie un certain nombre de liens potentiellement défectueux au lieu d'identifier exactement le lien défectueux.

Ce deuxième algorithme heuristique a été développé afin de cibler ou rétrécir la zone de recherche du lien défectueux. Il est applicable seulement quand il y a un seul lien défectueux dans tout le réticule. Il permet de localiser étroitement le lien défectueux avec le moindre coût. Il reçoit en paramètre les rectangles non fonctionnels et retourne un sous-ensemble de liens intercellulaires qui satisfassent à ces deux conditions :

Condition 1 : Appartenir à tous les rectangles non fonctionnels.

Condition 2 : Être caractérisé comme potentiellement défectueux.

En d'autres termes, l'algorithme HEURISTIC_2 identifie les liens potentiellement défectueux qui appartiennent à l'intersection de tous les rectangles trouvés non fonctionnels. La figure 3-8 illustre un exemple d'application de l'algorithme HEURISTIC_2 dans une grille de 6×6 cellules. La présence du lien défectueux (0,1) Sud rend trois rectangles (R1, R2 et R3) non fonctionnels (figure 3-8.a).

- Les liens qui sont en commun aux trois rectangles sont :

- Lien sortant Est de la cellule (0,5).
- Lien sortant Nord de la cellule (1,5).
- Lien sortant Nord de la cellule (1,4).
- Lien sortant Nord de la cellule (1,3).
- Lien sortant Sud de la cellule (0,1).
- Lien sortant Sud de la cellule (0,2).
- Lien sortant Sud de la cellule (0,3).

- Les liens qui sont en commun aux trois rectangles et qui sont potentiellement défectueux sont :

- Lien sortant Nord de la cellule (1,3).
- Lien sortant Sud de la cellule (0,1).

En effet, les cinq autres liens ont été caractérisés comme fonctionnels parce qu'ils appartiennent à d'autres rectangles fonctionnels :

- Le lien sortant Est de la cellule (0,5) a été caractérisé comme fonctionnel par plusieurs rectangles dont le rectangle vertical passant successivement par les cellules (0,5), (1,5), (2,5), (2,4), (2,3), (2,2), (2,1), (2,0), (1,0), (1,1), (1,2), (1,3), (1,4), (0,4).
- Le lien sortant Nord de la cellule (1,5) a été caractérisé comme fonctionnel par plusieurs rectangles dont le rectangle horizontal passant successivement par les cellules (0,5), (1,5), (1,4), (2,4), (3,4), (4,4), (5,4), (5,3), (4,3), (3,3), (2,3), (1,3), (0,3), (0,4).
- Le lien sortant Nord de la cellule (1,4) a été caractérisé comme fonctionnel par le rectangle horizontal passant successivement par les cellules (0,5), (1,5), (1,4), (1,3), (2,3), (3,3), (4,3), (5,3), (5,2), (4,2), (3,2), (2,2), (1,2), (0,2), (0,3), (0,4).
- Le lien sortant Sud de la cellule (0,2) a été caractérisé comme fonctionnel par le rectangle horizontal passant successivement par les cellules (0,5), (1,5), (1,4), (1,3), (2,3), (3,3), (4,3), (5,3), (5,2), (4,2), (3,2), (2,2), (1,2), (0,2), (0,3), (0,4).
- Le lien sortant Sud de la cellule (0,3) a été caractérisé comme fonctionnel par le chemin passant successivement par les cellules (0,5), (1,5), (1,4), (2,4), (3,4), (4,4), (5,4), (5,3), (4,3), (3,3), (2,3), (1,3), (0,3), (0,4).

L'appel de l'algorithme HEURISTIC_2 identifie donc deux liens (figure 3-8.b).

Comme le montre cet exemple, l'algorithme HEURISTIC_2 identifie un nombre faible de liens potentiellement défectueux parmi lesquels figure le lien défectueux recherché. Cet algorithme cible donc étroitement la recherche. Pour discerner exactement lequel de ces liens est le lien défectueux, on applique l'algorithme HEURISTIC_1. L'algorithme HEURISTIC_1 prend individuellement chacun des liens identifiés par HEURISTIC_2 et le connecte à TDI-TDO par des éléments fonctionnels (figure 3-8.c) et teste le chemin résultant. Dans l'exemple de la figure 3-7.c le chemin Ch1 est complet et fonctionnel alors le lien (1,3) Nord est fonctionnel. Par conséquent, le lien Sud sortant de la cellule (0,1) est le lien défectueux recherché. L'organigramme donné à la figure 3-9 montre les étapes pour localiser le seul lien défectueux présent dans le réticule. La figure 3-10 présente le pseudo code de l'algorithme de diagnostic d'un réticule à un seul lien défectueux.

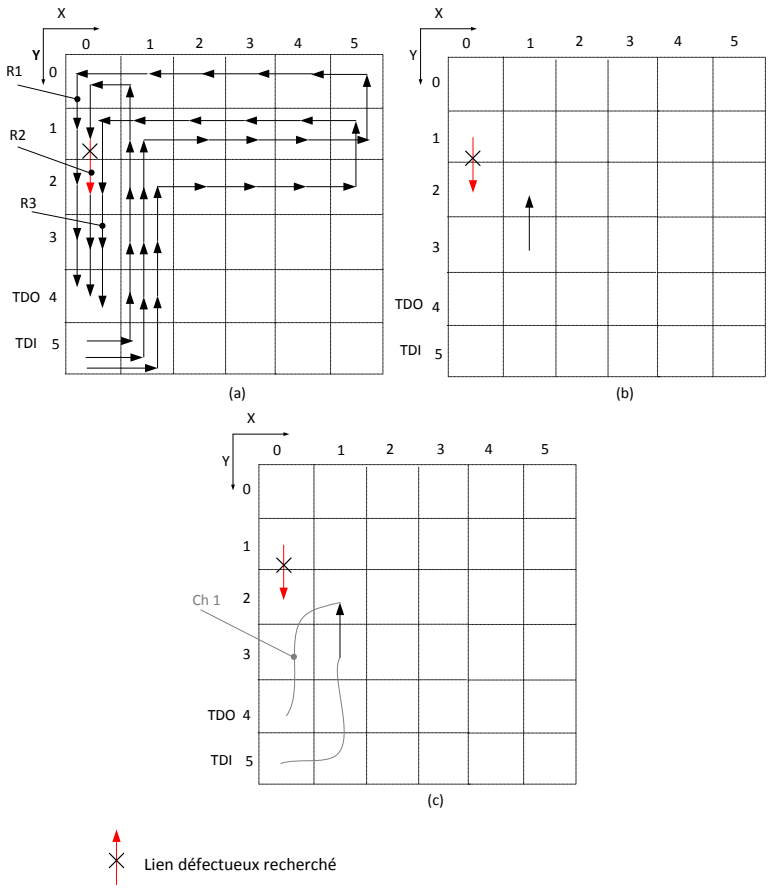


Figure 3-8: Identification d'un lien intercellulaire défectueux avec les algorithmes heuristiques (a) Rectangles non fonctionnels (b) Liens identifiés par l'algorithme HEURISTIC_2 (c) Application de l'algorithme HEURISTIC_1.

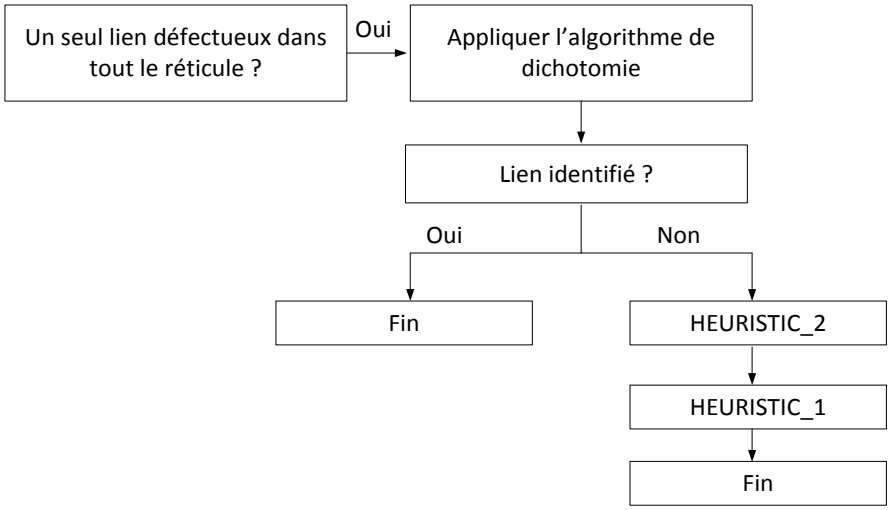


Figure 3-9: Recherche du seul lien défectueux du réticule.

Comme c'est mentionné à la figure 3-10, si le lien défectueux n'est pas précisément localisé après l'analyse des chemins rectangulaires non fonctionnels par l'algorithme de dichotomie, alors l'algorithme HEURISTIC_2 est appelé pour cibler la recherche de la défectuosité et déterminer le ou les liens avec une forte probabilité d'être défectueux. Afin de valider la fonctionnalité d'un plus grand nombre de liens possible, les chemins présents dans les listes *heuristic_list* et *unconnected_s_list* (s'ils existent) seront également testés. Les chemins de *heuristic_list* sont traités par l'algorithme HEURISTIC_1. Les segments *chemin* $_i$ de la liste *unconnected_s_list* sont d'abord complétés par des éléments fonctionnels pour avoir des chemins complets (p_i). Si p_i est fonctionnel alors tous les éléments de *chemin* $_i$ sont caractérisés comme fonctionnels sinon *chemin* $_i$ sera analysé par l'algorithme HEURISTIC_1 en testant ses liens un par un.

```
1:  Début
2 :  Configurer les rectangles;
3 :  Pour chaque Rectangle i faire
4 :    Si (TEST (Rectangle i) = vrai)
5 :    Caractériser les éléments de Rectangle i comme fonctionnels;
6 :    Sinon
7 :    Ajouter Rectangle i à nonfunctional_Rect_list;
8 :    Fin Si
9 :  Fin Pour
10:  Pour chaque Rectangle i de nonfunctional_Rect_list faire
11:     DICHOTOMIQUE (Rectangle i);
12:  Fin Pour
13:  Si le lien défectueux est identifié alors
14:          //valider la fonctionnalité de plus de liens possible//
15:          Si (heuristic_list.taille >0)
16:          HEURISTIC_1 (heuristic_list);
17:          Fin Si
18:          Si (unconnected_s_list.taille >0)
19:              Pour chaque chemin i de unconnected_s_list faire
20:              pi=CONNECT (chemin i, TDI,TDO);
21:              Si (pi est complet et TEST (pi)=vrai) alors
22:              Caractériser les éléments de chemin i comme fonctionnels;
23:              Sinon Si (pi est incomplet) alors
24:              Insérer (chemin i ,list2) ;
25:              HEURISTIC_1 (list2) ;
26:              Fin Si
27:             Fin Pour
28:           Fin Si
29:  Sinon
30:          //Cibler la recherche du lien défectueux //
31:          list1= HEURISTIC_2 (nonfunctional_Rect_list);
32:          HEURISTIC_1 (list1);
33:          //valider la fonctionnalité de plus de liens possible//
34:          Si (heuristic_list.taille >0)
35:          HEURISTIC_1 (heuristic_list);
36:          Fin Si
37:          Si (unconnected_s_list.taille >0)
38:              Pour chaque chemin i de unconnected_s_list faire
39:              pi=CONNECT (chemin i, TDI,TDO);
40:              Si (pi est complet et TEST (pi)=vrai) alors
41:              Caractériser les éléments de chemin i comme fonctionnels;
42:              Sinon Si (pi est incomplet) alors
43:              Insérer (chemin i ,list2) ;
44:              HEURISTIC_1 (list2) ;
45:              Fin Si
46:             Fin Pour
47:          Fin Si
48:   Fin Si
49:   Fin
```

Figure 3-10 : Pseudo code de l'algorithme de diagnostic dans le cas où un seul lien défectueux est présent dans le réticule.

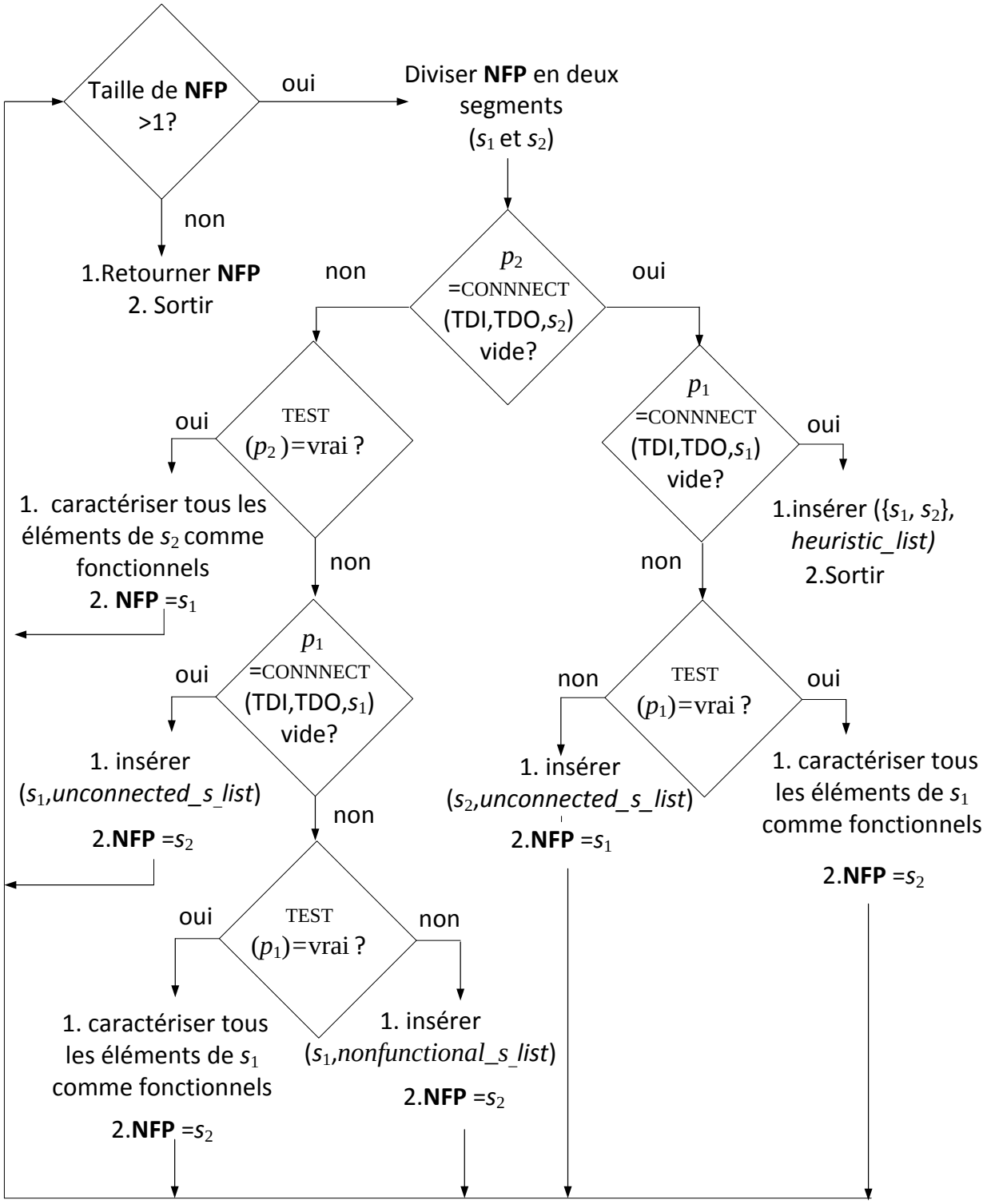


Figure 3-11: Organigramme de l'algorithme de dichotomie.

L'algorithme de dichotomie proposé dans ce mémoire localise dans la plupart des cas les liens défectueux présents dans le réticule. Dans le cas où il retourne un sous ensemble de liens potentiellement défectueux, l'algorithme HEURISTIC_1 est appliqué pour distinguer entre les liens fonctionnels et les liens défectueux de cet ensemble. Dans le cas d'un réticule à un seul lien défectueux, l'algorithme HEURISTIC_2 a été développé afin de cibler la recherche et déterminer les liens potentiellement défectueux les plus susceptibles de contenir la défectuosité. L'algorithme de dichotomie complété par les heuristiques augmente ainsi l'espace des éléments fonctionnels et par conséquent les éléments utilisables par la plateforme pour faire les interconnexions nécessaires entre les puces déposées sur sa surface. L'algorithme de dichotomie localise les liens défectueux même s'ils sont tous sur le même chemin ou sont immédiatement voisins. Dans le pire cas, quand il n'y a qu'un seul lien défectueux, l'algorithme de dichotomie est de complexité $O(\log_2 n)$ pour un chemin contenant n liens intercellulaires.

3.5 Routage basé sur l'algorithme de LEE

Afin de tester un segment donné il a été mentionné qu'il faut le compléter par les cellules liées physiquement aux ports TDI et TDO comme le montre la figure 3-3. Il faut donc développer un algorithme de routage qui trouve un chemin fonctionnel liant la cellule TDI à la première cellule du segment et la dernière cellule du segment à TDO. Il s'agit donc de trouver un chemin fonctionnel liant deux points donnés. Pour répondre à cet objectif, un algorithme basé sur le routage de LEE a été développé. La phase d'exploration dans l'algorithme de LEE peut être vue comme une propagation d'ondes depuis la cellule source jusqu'à la cellule destination. Dans l'algorithme de LEE standard [31], chaque cellule est marquée (indexée) par un numéro. La source est marquée par un '0' et le front d'onde se propage vers toutes les cellules immédiatement adjacentes à la source. Chaque cellule adjacente et qui est non bloquante sera marquée par '1'. Ensuite chaque cellule non bloquante et adjacente à une cellule marquée par '1' sera marquée par '2'. Et ainsi de suite. La phase d'exploration s'arrête quand la cellule destination est atteinte ou bien lorsque le front d'onde ne peut plus se propager. Il a été démontré que la complexité en temps et en espace de l'algorithme de LEE est de $O(H \times L)$ pour une grille de dimensions H×L.

L'algorithme de dichotomie proposé dans ce mémoire recherche la défectuosité au niveau des liens intercellulaires et non pas au niveau des cellules (l'état d'une cellule se déduit à partir de ses

liens). Par conséquent l'algorithme de LEE standard tel que décrit plus haut ne convient plus. Un algorithme de LEE modifié a été implémenté où au lieu que chaque cellule soit marquée par un seul index, elle est marquée par quatre index (West_in, East-in, South-in et North-in) comme le montre la figure 3-12.

- West_in représente l'index du lien intercellulaire entrant à la cellule et provenant de sa voisine Ouest.
- East_in représente l'index du lien intercellulaire entrant à la cellule et provenant de sa voisine Est.
- South_in représente l'index du lien intercellulaire entrant à la cellule et provenant de sa voisine Sud.
- North_in représente l'index du lien intercellulaire entrant à la cellule et provenant de sa voisine Nord.

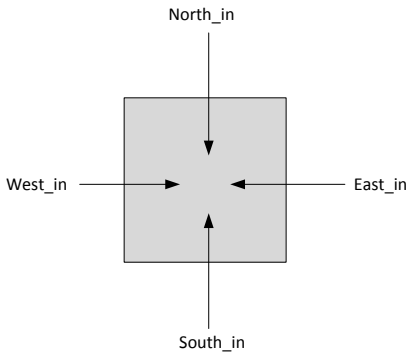


Figure 3-12: Cellule marquée par quatre index (North_in, West_in, South_in, et East_in).

Les étapes de la phase d'exploration de l'algorithme développé sont données à la figure 3-13. Une fois la cellule de destination est atteinte, la phase d'exploration s'arrête et le chemin est retracé à l'envers c'est-à-dire depuis la cellule destination vers la cellule source en suivant l'ordre décroissant du compteur.

La figure 3-14, illustre un exemple de la phase d'exploration de l'algorithme développé dans une grille bidimensionnelle de 6×6 cellules. Le but est de chercher un chemin fonctionnel entre les deux cellules Source (S) et Destination (T). Initialement la valeur du Compteur est égale à '1' et la liste est vide. Partant de la cellule S (*Cellule_Actuelle* = S). Supposons que le lien sortant de S

vers la direction Est et le lien sortant vers la direction Nord sont fonctionnels, alors le West_in de sa voisine Est et le South_in de sa voisine Nord sont marqués par '1'. Les voisines Est et Nord sont ajoutées à la liste. Le compteur augmente de 1. Chaque élément de cette liste devient à son tour la *Cellule_Actuelle* et ses liens sortants sont vérifiés. Si un lien est fonctionnel alors le lien en question sera marqué par la valeur du compteur et la cellule voisine dans la direction du lien sera ajoutée à la Liste. Et ainsi de suite, jusqu'à ce que la cellule (T) soit atteinte. Une fois que la cellule de destination est dans la liste, la phase d'exploration s'arrête et le chemin est retracé à l'envers (depuis la destination jusqu'à la source) en suivant l'ordre décroissant des West_in, East_in, South_in et North_in. L'algorithme de LEE permet de trouver un chemin liant deux points donnés si un tel chemin existe. Le chemin trouvé est le plus court possible.

```
1. Compteur = 1;
   Cellule_Actuelle = Source;
   Liste = Source;
   Voisine_EST   = la cellule voisine à Cellule_Actuelle dans la direction EST;
   Voisine_OUEST = la cellule voisine à Cellule_Actuelle dans la direction OUEST;
   Voisine_NORD  = la cellule voisine à Cellule_Actuelle dans la direction NORD;
   Voisine_SUD   = la cellule voisine à Cellule_Actuelle dans la direction SUD;

2.  Si le lien EST de Cellule_Actuelle est fonctionnel alors
        West_in de Voisine_EST = Compteur;
        Insérer (Voisine_EST, Liste);
    Si le lien OUEST de Cellule_Actuelle est fonctionnel alors
        East_in de Voisine_OUEST = Compteur;
        Insérer (Voisine_OUEST, Liste);
    Si le lien NORD de Cellule_Actuelle est fonctionnel alors
        South_in de Voisine_NORD = Compteur;
        Insérer (Voisine_NORD, Liste);
    Si le lien SUD de Cellule_Actuelle est fonctionnel alors
        North_in de Voisine_SUD = Compteur;
        Insérer (Voisine_SUD, Liste);

3. Retirer Cellule_Actuelle de Liste;
4. Compteur ++,
5. Si (la destination est dans Liste)
   alors retracer le chemin
       Quitter;
   Sinon
     Pour chaque Cellule i de Liste faire
       Cellule_Actuelle = Cellule i;
       Répéter les étapes 2,3,4,5;
     Fin Pour
```

Figure 3-13: Pseudo code de l'algorithme de LEE développé.

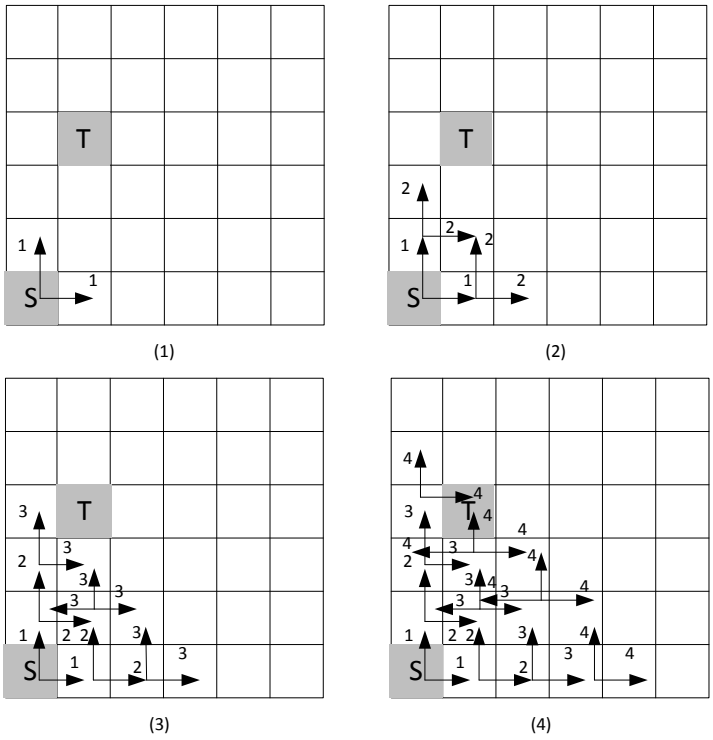


Figure 3-14: Exemple de la phase d'exploration dans l'algorithme de LEE développé.

3.6 Sommaire

Ce chapitre a décrit les algorithmes de recherche de(s) élément(s) défectueux se trouvant dans un chemin non fonctionnel. En effet, dire qu'un chemin est non fonctionnel revient à dire que toutes ses cellules et tous ses liens sont des ressources inutilisables c'est-à-dire que la chaîne JTAG nécessaire pour la configuration du WaferIC ne doit pas passer par ces ressources. Afin d'augmenter l'espace des éléments fonctionnels et par conséquent les éléments utilisables, l'algorithme de dichotomie a été implémenté. Il reçoit en argument le chemin non fonctionnel et retourne le lien défectueux se trouvant sur ce chemin. Comme le principe de la dichotomie l'indique, le chemin non fonctionnel est divisé en deux segments puis chaque segment est testé. Le segment trouvé non fonctionnel est divisé à son tour et ainsi de suite. Ceci se répète d'une façon itérative jusqu'à arriver à la défectuosité. Le fait de tester les deux segments d'un chemin aide à localiser tous les liens défectueux qui peuvent se trouver sur ce chemin même si les liens défectueux sont immédiatement voisins. Cela permet aussi d'augmenter le nombre de liens caractérisés comme fonctionnels. Chaque segment à tester doit être complété par des liens

fonctionnels pour former un chemin complet à travers lequel des flux de bits peuvent être envoyés. Un algorithme basé sur le routage de LEE a été développé pour répondre à cet objectif. Trouver des liens déjà caractérisés comme étant fonctionnels pour compléter un segment donné constitue la partie la plus difficile de l'algorithme. En effet, il arrive dans certains cas qu'aucun des deux segments ne puisse être complété par les cellules TDI-TDO. On se retrouve avec plusieurs liens non testés. À ce moment un algorithme heuristique est appliqué. Il prend les liens non testés un par un et les lie aux cellules TDI-TDO (par des éléments fonctionnels) afin de les tester. Quand un lien particulier ne peut pas être lié à TDI-TDO il reste dans un état indéterminé. Il a été montré dans ce chapitre que dans le cas où il n'y a qu'un seul lien défectueux dans tout le réticule, il est préférable d'utiliser un algorithme heuristique basé sur le principe de l'intersection qui cible la recherche et retourne un ensemble très restreint de liens potentiellement défectueux susceptibles de contenir la défectuosité. Cet algorithme retourne les liens potentiellement défectueux qui sont communs à tous les rectangles non fonctionnels.

Le chapitre suivant décrit les résultats donnés par l'algorithme de diagnostic. La première partie estime le nombre de défectuosités que peut contenir un réticule. Cette estimation est faite en se basant sur deux lois probabilistes (La loi de Poisson et la loi binomiale négative). La deuxième partie donne les résultats de simulation dans le cas d'un seul lien défectueux et dans le cas de plusieurs liens défectueux. La troisième partie présente les résultats expérimentaux extraits à partir d'un prototype du WaferIC.

CHAPITRE 4 RÉSULTATS

4.1 Introduction

Ce chapitre présente les résultats obtenus de l'application de l'algorithme de diagnostic développé dans le cadre de ce projet de maîtrise. L'algorithme a été développé sous Linux en utilisant le langage de programmation C++ sous l'environnement de développement QT Creator et avec le compilateur Gcc. Le code comporte plus de 3200 lignes. Quatre classes principales ont été construites. Une classe définit une cellule, une deuxième classe définit la zone sur laquelle le diagnostic sera fait c'est-à-dire une matrice de 32×32 cellules (la taille est générique) et définit aussi les liens qui sont non « diagnosticables » ainsi que les liens qui n'existent pas comme les liens sortants de la matrice. La troisième classe contient les fonctions concernant le diagnostic comme la création des rectangles, l'algorithme de dichotomie, l'algorithme de LEE ainsi que les deux algorithmes heuristiques. Une dernière classe concerne le test, elle contient les fonctions qui activent les chemins et injectent des liens défectueux. Afin de vérifier le fonctionnement de cet algorithme par simulation, le nombre théorique de défectuosités que peut contenir un réticule du WaferIC est estimé. Plusieurs méthodes mathématiques existent pour estimer ou prévoir au mieux le nombre de défectuosités possibles dans un circuit intégré donné. Les lois les plus usuelles sont la loi de Poisson et la loi binomiale négative, qui sont des distributions de probabilité discrètes. La première section de ce chapitre présente une estimation théorique du nombre d'éléments défectueux présents dans un réticule du WaferIC en se basant sur ces deux lois probabilistes. Le chapitre contient ensuite deux sections présentant les résultats obtenus par simulation et les résultats expérimentaux faits sur un réticule d'un mini prototype du WaferIC conçu pour émuler le comportement de base du circuit.

4.2 Estimation du nombre de défectuosités dans un réticule par la loi de Poisson et la loi binomiale négative

Le nombre de défectuosités que peut contenir la chaîne JTAG dans un réticule du WaferIC peut être estimé à l'aide des lois probabilistes. Dans les premiers jours de la microélectronique, la loi de Poisson était utilisée pour estimer le nombre de défectuosités dans un circuit intégré en utilisant la relation suivante :

$$P(x) = \frac{e^{-\lambda}}{x!} \lambda^{x} \quad (7)$$

où *x* est le nombre de défectuosités que peut contenir le circuit intégré, λ une constante donnant le nombre moyen de défectuosités et P(*x*) est la probabilité que le circuit contienne *x* défectuosités.

La loi de Poisson est une loi uniforme qui considère que toutes les parties du circuit ont la même probabilité de contenir des défectuosités. Or plus tard, il s'est avéré que ceci n'est pas totalement vrai puisque les défectuosités avaient tendance à se grouper. Donc certaines parties du circuit avaient plus de probabilité de contenir des défectuosités que d'autres. Comme solution à ce problème, des modèles ont été développés à partir de la loi de Poisson. Ces modèles ont abouti à la fin à la loi binomiale négative définie par la relation suivante :

$$P(x) = \frac{\Gamma(\alpha+x)}{x!\,\Gamma(\alpha)} \times \frac{(\lambda/\alpha)^{x}}{(1+\frac{\lambda}{\alpha})^{\alpha+x}} \quad (8)$$

où *x* est le nombre de défectuosités que peut contenir le circuit, λ est le nombre moyen de défectuosités dans le circuit, P(*x*) est la probabilité que le circuit contienne *x* défectuosités, Γ est la fonction gamma définie par $\Gamma(x) = (x-1)!$ et α est le facteur de groupement. Plus le facteur de groupement α est grand, moins les défectuosités sont groupées. Lorsque α tend vers l'infini, la loi binomiale négative rejoint la loi de Poisson.

En se basant sur ces deux lois de probabilité, il serait possible d'estimer le nombre de défectuosités qu'une chaîne JTAG, dans un réticule, est susceptible de contenir. Un facteur indispensable doit toutefois être calculé en premier lieu qui est le nombre moyen λ de défectuosités. Pour cela, on dispose des données suivantes :

- Densité typique de défectuosités = 1 défectuosité par 5 cm^2 [9];
- Surface d'une cellule = 0.56mm × 0.56mm = 0.3136 mm^2 [32];
- Surface d'un *NanoPad* = 0.00847 mm^2 [32];
  - → Surface utilisée pour la logique et les interconnexions = 0.3136 – (16 × 0.00847)= 0.17808 mm^2.
- D'après [21], 5.2% de la logique dans la cellule est utilisée par la chaîne JTAG.

→ Surface des éléments JTAG au sein d'une cellule = 0.17808 × 5.2% =0.0092 mm^2=0.000092 cm^2.

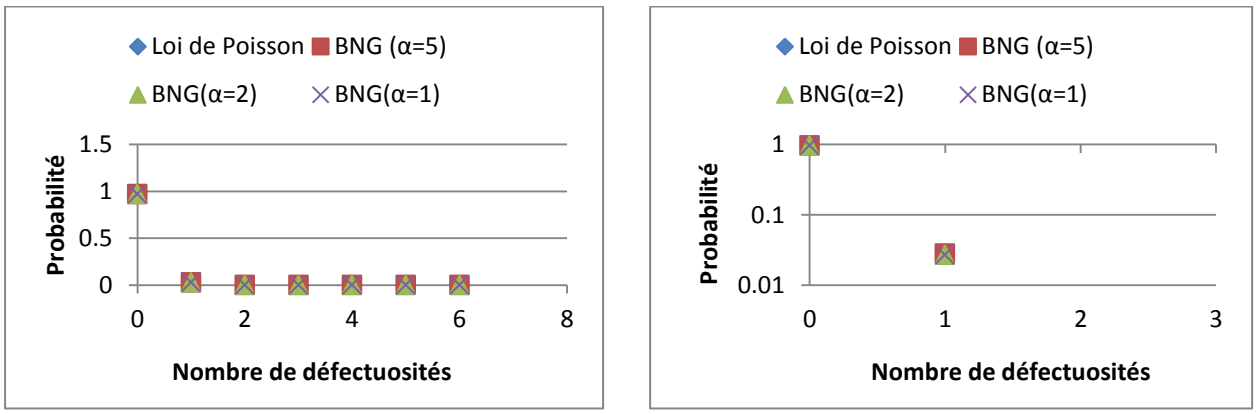
→ Surface des éléments JTAG au sein d'un réticule = 0.000092 cm^2 × 1024=0.094 cm^2.

Ainsi le nombre moyen de défectuosités λ par réticule, égal à la surface du circuit × densité de défectuosités, est alors :

$$\lambda = 0.094\ cm^2 \times 1\ \text{défectuosité}/5\ cm^2 = 0.018\ \text{défectuosités par réticule} \quad (9)$$

Étant donné ce faible nombre de défectuosités par réticule, il est inutile de prendre en considération le groupement des défectuosités. Par conséquent, la loi de Poisson est suffisante pour modéliser les défectuosités qui peuvent être présentes dans un réticule. La loi binomiale négative est toutefois utilisée pour confirmer ces résultats obtenus par la loi de Poisson.

La loi binomiale négative tient compte du groupement des défectuosités défini dans la variable α. Selon [29], les valeurs moyennes de α sont comprises dans l'intervalle [0.5, 5]. Le nombre de défectuosités dans un réticule est estimé dans ce qui suit avec la loi binomiale négative pour α égal à 5, 2 et 1.



(a) (b)

Figure 4-1: Estimation du nombre de défectuosités par la loi de Poisson et la loi binomiale négative pour λ=0.018 (a) L'axe des ordonnées en échelle linéaire (b) L'axe des ordonnées en échelle logarithmique.

On remarque à travers la distribution de Poisson et la distribution binomiale négative, données à la figure 4-1, que la probabilité qu'il n'y ait aucune défectuosité dans la chaîne JTAG au sein du réticule de 32×32 cellules est toujours la plus élevée (≈0.99). La probabilité pour que la chaîne

contienne une défectuosité ou plus est relativement faible. Estimer le nombre de défectuosités dans la chaîne est important pour pouvoir caractériser le comportement de l'algorithme de diagnostic. En effet, dans le paragraphe suivant (4.3) on montrera que pour simuler le comportement de l'algorithme, un certain nombre de défectuosités sera injecté aléatoirement dans le modèle abstrait du réticule. L'estimation du nombre de défectuosités que la chaîne JTAG dans un réticule est susceptible de contenir, permet de donner une idée sur le nombre de défectuosités à injecter.

4.3 Résultats de simulation

L'algorithme de diagnostic a été développé avec le langage C++ en utilisant le logiciel QT Creator sous le compilateur Gcc. Le pseudo code décrivant l'environnement de simulation est donné à la figure 4-2. Pour simuler l'algorithme, une carte *m_linkDefects* contenant quelques liens aléatoires défectueux est créée. Pour une matrice de 32×32 cellules, 31 rectangles verticaux puis 31 rectangles horizontaux sont créés avec la fonction CREATE_RECTANGULAR_PATH pour couvrir les liens et les cellules du réticule. Chaque rectangle créé est testé par la fonction TEST. Tester un rectangle par simulation est fait en comparant ses liens avec ceux présents dans la « carte » *m_linkDefects*. Si un lien du rectangle est présent dans *m_linkDefects* alors le rectangle est non fonctionnel, sinon il est fonctionnel. Si un rectangle est fonctionnel alors toutes ses cellules et tous ses liens sont caractérisés comme fonctionnels. Mais si le rectangle est non fonctionnel alors il sera ajouté à la liste *nonfunctional_Rect_list*. Chaque rectangle de cette liste est analysé par les algorithmes de diagnostic pour localiser le(s) élément(s) défectueux.

```
1:  Début
2:  Remplir m_linkDefects;
3:  i=0;
4:  Tant que (i < 62)
5:        Rect= CREATE_RECTANGULAR_PATHS;
6:        i++;
7:           Si (TEST (Rect) =vrai)
8:           Caractériser les éléments de (Rect) comme fonctionnels;
9:           Sinon
10:          insérer (Rect,nonfunctional_Rect_list);
11:          Fin si
12: Fin tant que
13: Pour chaque Rect i dans nonfunctional_Rect_list faire
14:   DIAGNOSTIC (Rect i);
15: Fin pour
16: Fin
```

Figure 4-2: Pseudo code de l'environnement de simulation.

4.3.1 Comportement de l'algorithme en présence d'un seul lien défectueux

Pour simuler le comportement de l'algorithme de diagnostic en présence d'un seul lien défectueux, un lien est inséré dans la carte *m_linkDefects* qui est une paire de paramètres. Le premier est l'identifiant appelé aussi clé et le deuxième est la donnée. La carte associe une clé à une donnée. La clé de la carte est les coordonnées d'une cellule (x,y) alors que la donnée est la direction du lien sortant de cette cellule. Puisque le réticule contient 3843 liens diagnosticables donc 3843 tests ont été effectués où à chaque test un lien est injecté dans *m_linkDefects* comme étant un lien défectueux. Comme le montre le tableau 4.1, dans 3808 tests (99.08%) l'algorithme de dichotomie identifie exactement le lien défectueux inséré et dans 35 tests (0.91%) l'algorithme de dichotomie n'est pas capable de localiser le lien défectueux. Les 35 liens non trouvés sont montrés dans la figure 4-3 : Un lien de la figure 4-3 appartient à plusieurs chemins rectangulaires à la fois, et par conséquent lorsqu'il est défectueux, il rend un grand nombre de chemins rectangulaires non fonctionnels. Ainsi, la fonction CONNECT de l'algorithme de dichotomie ne trouve pas assez de liens caractérisés comme fonctionnels pour lier des segments à TDI-TDO afin de les tester. Pour localiser les liens de la figure 4-3, les algorithmes HEURISTIC_2 puis HEURISTIC_1 sont appliqués. On remarque que les liens montrés à la figure 4-3, sont situés à la frontière du réticule, il serait donc facilement localisés en étendant le diagnostic sur deux réticules voisins au lieu que ça soit sur un seul réticule.

Tableau 4.1 Résultats donnés par l'algorithme de dichotomie pour un seul lien défectueux injecté (nombre de liens total dans le réticule est égal à 3843).

Type de liens	Nombre	Pourcentage
Liens localisés par l'algorithme de dichotomie	3808	99.08 %
Liens non localisés par l'algorithme de dichotomie	35	0.91 %

Lorsque l'un des 35 liens montrés dans la figure 4-3 est défectueux, l'algorithme de dichotomie retourne un grand nombre de liens (Tableau 4.2, colonne 2). Pour cibler la recherche et identifier les liens les plus susceptibles d'être défectueux, l'algorithme HEURISTIC_2 est appliqué. Il identifie les liens intercellulaires potentiellement défectueux qui sont en commun à tous les rectangles trouvés non fonctionnels. Le tableau 4.2 (colonne 3) montre les liens identifiés par

l'algorithme HEURISTIC_2. On remarque que parmi les liens identifiés, figure le lien défectueux injecté et un ou deux autres liens. Pour encore cibler la recherche ou identifier exactement le lien défectueux, l'algorithme HEURISTIC_1 est appliqué sur les liens identifiés par HEURISTIC_2. Le tableau 4.2 (colonne 4) montre les résultats donnés après l'appel à l'algorithme HEURISTIC_1.

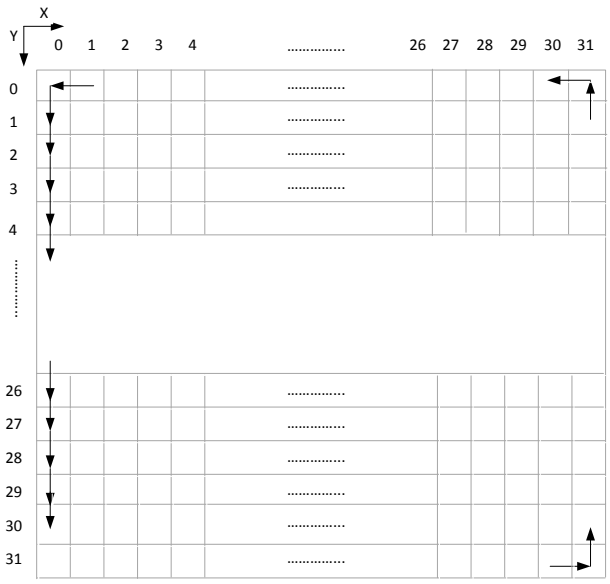


Figure 4-3: Liens non localisés par l'algorithme de dichotomie.

Tableau 4.2 Résultats correspondant aux liens de la figure 4-3 (nombre total des liens du réticule est égal à 3843).

Lien défectueux injecté	Nombre de liens retournés par l'algorithme de dichotomie	Liens identifiés par l'algorithme HEURISTIC_2	Liens identifiés par les algorithmes HEURISTIC_2 et HEURISTIC_1	Nombre de liens potentiellement défectueux	Nombre de liens fonctionnels
(0,0,S)	33	(1,2,N), (1,0,O), (0,0,S)	(1,0,O), (0,0,S)	4	3839
(0,1,S)	66	(1,3,N), (0,1,S)	(0,1,S)	8	3835
(0,2,S)	99	(1,4,N), (0,2,S)	(0,2,S)	12	3831
⋮	⋮	⋮	⋮	⋮	⋮
(0,28,S)	1048	(1,30,N), (0,28,S)	(0,28,S)	116	3727
(0,29,S)	1080	(1,31,N), (0,29,S)	(0,29,S)	120	3723
(1,0,O)	33	(0,0,S), (1,0,O), (1,2,N)	(0,0,S), (1,0,O)	4	3839
(31,0,O)	6	(31,0,O), (31,1,N)	(31,0,O), (31,1,N)	5	3838
(31,1,N)	6	(31,0,O), (31,1,N)	(31,0,O), (31,1,N)	5	3838
(31,31,N)	34	(30,31,E), (31,31,N)	(30,31,E), (31,31,N)	3	3839
(30,31,E)	34	(30,31,E), (31,31,N)	(30,31,E), (31,31,N)	3	3839

*(x,y,d) : (x,y) sont les coordonnées de la cellule, d est la direction du lien sortant de la cellule (S=Sud, N=Nord, O=Ouest, E=Est).

La colonne 5 du tableau 4.2, donne le nombre de liens caractérisés comme potentiellement défectueux après l'application de l'algorithme de diagnostic donné à la figure 3-10. La colonne 6 du tableau donne le nombre de liens caractérisés comme fonctionnels par l'algorithme de diagnostic. On remarque qu'après l'application de l'algorithme de dichotomie, le nombre de liens potentiellement défectueux est trop élevé (colonne 2 du tableau 4.2). Ce nombre diminue à la fin de l'algorithme de diagnostic (colonne 5 du tableau 4.2). Ceci est dû au fait que les segments se trouvant dans les listes *heuristic_list* et *unconnected_s_list* sont analysés afin de valider la fonctionnalité de leurs liens. En effet, tel que montré dans le pseudo code de la figure 3-10 (ligne 33), après que les algorithmes heuristiques HEURISTIC_2 et HEURISTIC_1 soient appliqués pour cibler la recherche du lien défectueux, celui-ci est localisé. Cependant, il faut valider la fonctionnalité des liens constituant les segments des listes *heuristic_list* et *unconnected_s_list*. Il faut donc vérifier que ces liens appartiennent à des chemins fonctionnels. Les segments de *heuristic_list* sont traités par l'algorithme HEURISTIC_1 en complétant chaque lien par des liens fonctionnels et testant le chemin résultant. Chaque segment $chemin_i$ de la liste *unconnected_s_list* doit être complété par des liens fonctionnels afin d'avoir un chemin complet p_i. Si le chemin p_i est fonctionnel alors tous les liens du segment $chemin_i$ sont déclarés fonctionnels. Si aucun chemin complet p_i couvrant le segment $chemin_i$ alors ce dernier sera analysé par l'algorithme HEURISTIC_1 en testant ses liens un par un.

On remarque que dans le pire cas et lorsque le réticule contient un seul lien défectueux, le nombre de liens fonctionnels localisés par l'algorithme de diagnostic est égal à 3723 d'un total de 3843 soit un pourcentage de 96.87%. Le pourcentage des liens potentiellement défectueux est égal à 3.12%. Chacun de ces liens reste dans l'état potentiellement défectueux parce qu'il est impossible de trouver un chemin le liant à TDI-TDO et constitué seulement de liens fonctionnels. Valider la fonctionnalité de ces liens n'est pas possible avec l'algorithme développé. En effet, l'algorithme de LEE adopté pour connecter un lien donné à TDI-TDO utilise uniquement des liens caractérisés comme fonctionnels. Cependant, une modification au niveau de l'algorithme de LEE tolérant l'utilisation des liens dont la fonctionnalité n'est pas garantie pour connecter un lien potentiellement défectueux à TDI-TDO, pourrait réduire le nombre de liens potentiellement défectueux et par conséquent augmenter celui des liens fonctionnels.

L'algorithme de dichotomie complété par les heuristiques HEURISTIC_1 et HEURISTIC_2 permet de localiser presque tout lien intercellulaire défectueux présent dans le réticule (Tableau 4.2, colonne 4). On remarque à travers le tableau 4.2 qu'en appliquant l'algorithme HEURISTIC_1 suite à l'algorithme HEURISTIC_2, le lien défectueux recherché est exactement localisé sauf pour six liens qui sont (0,0) Sud, (1,0) Ouest, (31,0) Ouest, (31,1) Nord, (31,31) Nord, et (30,31) Est. Ces liens sont localisés dans les coins du réticule. Dans ces cas, deux liens sont identifiés par l'algorithme HEURSTIC_1 parmi lesquels figure le lien défectueux recherché. L'exemple donné ci-dessous est un cas de figure expliquant pourquoi les liens situés dans le coin n'arrivent pas être exactement localisés par l'algorithme HEURISTIC_1.

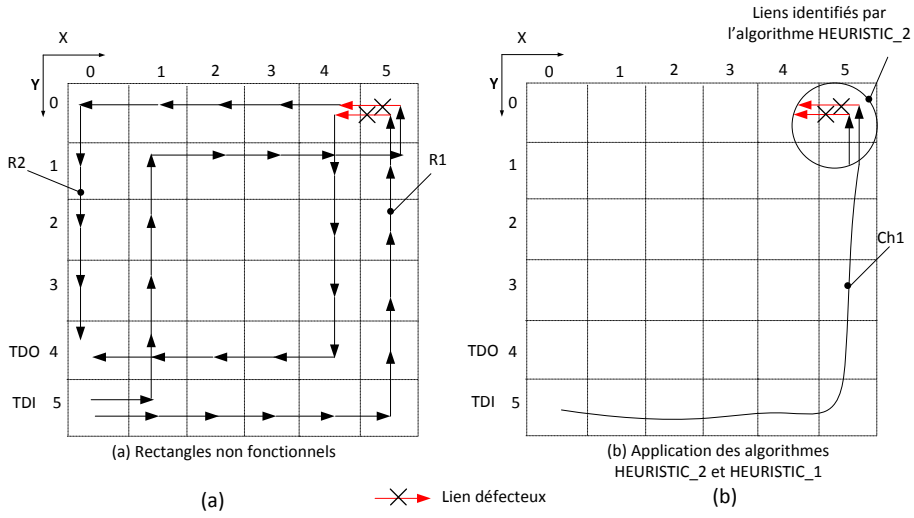


Figure 4-4: Exemple d'application des algorithmes heuristiques pour localiser un lien défectueux appartenant aux liens de la figure 4-3 et situé dans le coin du réticule (a) rectangles non fonctionnels (b) application de l'algorithme HEURISTIC_1 sur les liens identifiés par l'algorithme HEURISTIC_2.

Supposant que le lien intercellulaire Ouest sortant de la cellule (5,0) est défectueux. Le rectangle vertical R1 ainsi que le rectangle horizontal R2 ayant ce lien en commun sont donc non fonctionnels (figure 4-4.a). Tous les autres rectangles de la grille sont fonctionnels. L'application de l'algorithme HEURISTIC_2 retourne les liens qui sont en communs aux deux rectangles R1 et R2 mais aussi qui n'ont pas été caractérisés comme fonctionnels par d'autres rectangles.

Application de l'algorithme HEURISTIC_2

Les liens qui sont en commun à R1 et R2 sont : (0,5) Est, (5,1) Nord, (5,0) Ouest.

Les liens qui sont en commun à R1 et R2 et qui n'ont pas été préalablement caractérisés comme fonctionnels sont : (5,1) Nord, (5,0) Ouest. En effet, le lien (0,5) Est appartient à plusieurs rectangles fonctionnels dont le rectangle vertical parcourant successivement les cellules (0,5), (1,5), (1,4), (1,3), (1,2), (1,1), (1,0), (0,0), (0,1), (0,2), (0,3) et (0,4).

Ainsi l'algorithme HEURISTIC_2 identifie les liens (5,1) Nord et (5,0) Ouest comme les liens parmi lesquels figure le lien défectueux.

Afin d'identifier exactement lequel des liens retournés par HEURISTIC_2 est le lien défectueux recherché, l'algorithme HEURISTIC_1 est appliqué. Il crée deux chemins complets chacun couvrant un lien.

Application de l'algorithme HEURISTIC_1

Couverture du lien (5,1) Nord par un chemin complet :

Par exemple, le chemin (Ch1) tel que montré dans la figure 4-4.b peut être utilisé pour lier le lien (5,1) Nord à la cellule TDI. Cependant aucun chemin constitué uniquement de liens fonctionnels ne peut être trouvé pour lier le lien (5,1) Nord à la cellule TDO. En effet, tout chemin liant le lien (5,1) Nord à TDO doit impérativement passer par le lien (5,0) Ouest qui est identifié comme potentiellement défectueux.

Couverture du lien (5,0) Ouest par un chemin complet :

Aucun chemin constitué uniquement de liens fonctionnels liant la cellule TDI au lien (5,0) Ouest ne peut être trouvé. En effet, tout chemin liant TDI à ce lien doit impérativement passer par le lien (5,1) Nord qui est identifié comme potentiellement défectueux.

Puisqu'aucun chemin complet couvrant (5,0) Ouest et aucun chemin complet couvrant (5,1) Nord n'ont pu être trouvés, les liens potentiellement défectueux identifiés par l'algorithme HEURISTIC_1 sont les mêmes que ceux identifiés par l'algorithme HEURISTIC_2. Il est

important à mentionner que ces cas limites (liens se trouvant à la frontière et dans les coins) peuvent être facilement résolus en jumelant l'analyse sur deux réticules côte à côte.

Limitation de l'algorithme de diagnostic

Comme l'indique la colonne 5 du tableau 4.2, lorsque l'un des liens montrés dans la figure 4-3 est défectueux, un ensemble de liens sont déclarés potentiellement défectueux par l'algorithme de diagnostic. Prenant l'exemple de la figure 4-5. Supposons que le lien défectueux est le lien sud sortant de la cellule de coordonnées (0,1). Tous les liens montrés dans la figure 4-5.a sont déclarés potentiellement défectueux par l'algorithme de diagnostic. En effet, il est impossible de tester les liens (1,2) Nord, (1,1) Nord, (1,0) Ouest, (0,0) Sud, (0,1) Sud et (1,1) Ouest (liens colorés en noir) puisqu'aucun chemin complet constitué uniquement de liens fonctionnels ne peut être trouvé pour couvrir un de ces liens cités. Cependant, ce n'est pas le cas pour les liens (1,2) Est et (1,1) Est (colorés en bleu). En effet, un chemin complet passant uniquement par des liens fonctionnels peut bien couvrir le lien (1,2) Est comme l'exemple donné à la figure 4-5.b. De même pour le lien (1,1) Est. Or l'algorithme de LEE est, par définition, un algorithme qui cherche le chemin fonctionnel le plus court entre deux points donnés. Donc pour lier le lien (1,2) Est à TDI, l'algorithme de LEE crée le chemin « Ch » montré dans la figure 4-5.c, car c'est bien le chemin fonctionnel le plus court. Quand il en vient à lier le lien (1,2) Est à TDO, aucun chemin fonctionnel ne peut être trouvé sans avoir à traverser une des cellules de « Ch » deux fois. Par conséquent, l'algorithme de diagnostic ne trouvera aucun chemin couvrant le lien (1,2) Est. Ce lien sera alors déclaré comme potentiellement défectueux. Même chose pour le lien (1,1) Est. Ceci représente une limitation de l'algorithme de diagnostic. Pour pallier ce problème, on peut toujours recréer le chemin, liant le lien (1,2) Est à TDI-TDO, en commençant d'abord par créer le segment qui le lie à TDO puis ensuite créer le segment qui le lie à TDI.

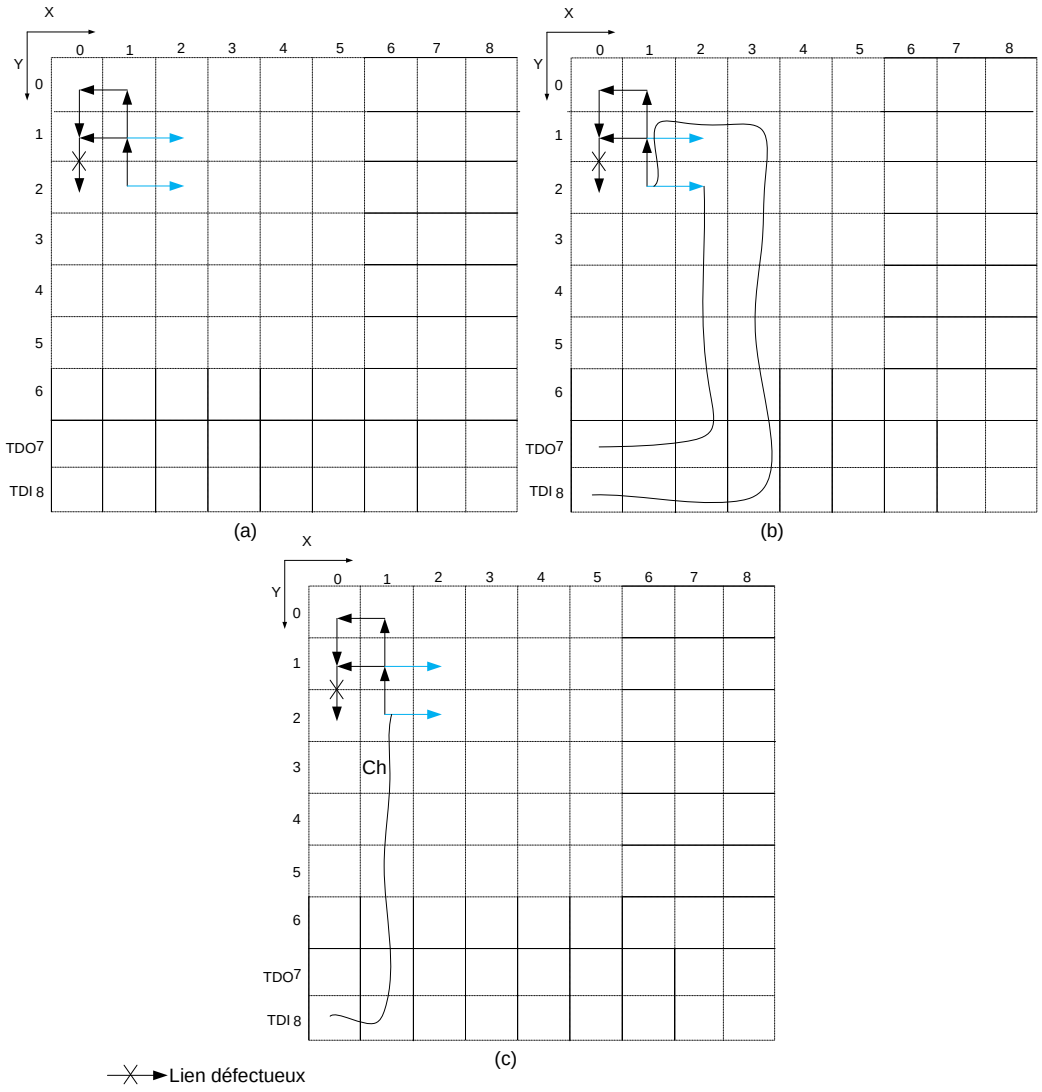


Figure 4-5 Exemple illustrant une limite de l'algorithme de LEE (a) liens déclarés potentiellement défectueux par l'algorithme de LEE (b) Exemple de chemin complet constitué uniquement de liens fonctionnels et couvrant le lien (1,2) Est (c) chemin créé par l'algorithme de LEE, liant la cellule TDI au lien (1,2) Est.

4.3.2 Comportement de l'algorithme en présence de plusieurs liens défectueux

En se basant sur la première section de ce chapitre, on a vu que la probabilité d'avoir plus d'une défectuosité sur les éléments de la chaîne JTAG est relativement très faible. Toutefois, simuler le comportement de l'algorithme en présence de plusieurs liens défectueux a été jugé intéressant. La figure 4-6 présente le comportement de l'algorithme de diagnostic si plusieurs liens défectueux sont injectés. Pour cela 2, 3, 4, 5, 6, 7 et 8 liens défectueux ont été aléatoirement injectés. Pour

chacun de ces nombres de liens défectueux, 100 essais ont été faits.L'annexe B présente un exemple (un cas de figure) où l'algorithme de dichotomie n'arrive pas à localiser des liens défectueux. On remarque à travers la figure 4-6 que l'algorithme de dichotomie est capable à lui seul de localiser précisément les 8 liens défectueux d'un réticule avec un taux de succès de 71%.

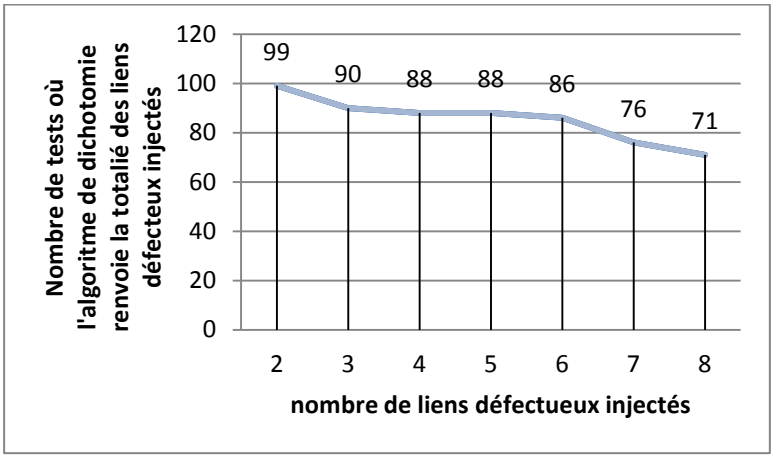


Figure 4-6: Performances de l'algorithme de dichotomie dans le cas où plusieurs liens défectueux sont injectés dans le réticule.

Analyse des résultats de simulation

On remarque à travers la figure 4-6 que dans la majorité des essais, l'algorithme de dichotomie réussit à localiser la totalité des liens défectueux injectés. Dans quelques essais, seulement quelques liens parmi les liens défectueux injectés sont localisés par l'algorithme de dichotomie. Pour le reste, il faut appliquer l'algorithme HEURISTIC_1 pour localiser le lien défectueux ou réduire la zone qui le délimite. L'appel ou non à l'algorithme heuristique pour localiser les défectuosités, dépend de deux facteurs :

- L'emplacement des liens défectueux : si parmi les défectuosités figure un des liens identifiés à la figure 4-3, l'algorithme de dichotomie risque de ne pas localiser précisément la totalité des liens défectueux. En effet, comme le montre l'exemple de l'annexe B, une défectuosité au niveau d'un ou de plusieurs liens montrés à la figure 4-3 cause la non-fonctionnalité d'un grand nombre de rectangles.
- Le nombre de liens défectueux : plus le nombre de défectuosités augmente, plus le nombre de chemins non fonctionnels augmente. Ainsi, le nombre de liens caractérisés comme

fonctionnels diminue. Et par conséquent l'algorithme de LEE risque de ne pas trouver des éléments fonctionnels pour compléter un segment donné par TDI-TDO.

4.4 Résultats expérimentaux

L'algorithme a été mis à profit par le test physique sur un réticule d'un prototype miniaturisé du WaferIC nommé mini-WaferIC. Le mini-WaferIC est constitué de 4 réticules (matrice de 2×2). La figure 1-16 du paragraphe 1.5.3 (Chapitre 1) illustre le montage nécessaire fait par l'équipe pour entrer en communication avec un réticule du mini-WaferIC. Ce montage a été utilisé pour nos tests.

Au total 62 flux de bits ont été envoyés vers le FPGA pour créer 31 rectangles verticaux puis 31 rectangles horizontaux. Au total 1 259 592 bits ont été injectés pour configurer les 62 rectangles. Quatre autres flux de données ont été injectés pour couvrir 100% des liens « diagnosticables » du réticule. Des données sont injectées à travers les chemins créés. Parmi les 62 rectangles, seulement six n'ont pas conduit le signal jusqu'au port de sortie TDO (Figure 4-7.a). Ces six rectangles sont donc non fonctionnels et chacun contient un ou plusieurs éléments défectueux (cellules ou liens intercellulaires). L'algorithme de dichotomie a donc été appliqué sur chacun de ces rectangles pour localiser les défectuosités. Chaque rectangle est divisé en deux segments. Le segment trouvé non fonctionnel est divisé à son tour en deux et ainsi de suite. Pour tester un segment donné, il faut relier sa première cellule à TDI et sa dernière cellule à TDO pour obtenir un chemin complet. L'algorithme de dichotomie s'arrête de diviser le chemin non fonctionnel lorsqu'il n'est plus en mesure de trouver des cellules et des liens fonctionnels pour connecter ses deux segments à TDI et TDO. À ce stade, un grand nombre d'éléments restent dans un état inconnu (Figure 4-7.b) et ils sont déclarés « potentiellement défectueux ». Pour réduire le nombre de liens potentiellement défectueux et pour discerner le plus possible la zone contenant la défectuosité, l'algorithme heuristique HEURISTIC_1 a été appliqué. Cet algorithme a créé plusieurs chemins, chacun couvre un lien potentiellement défectueux de la zone obtenue dans la figure 4-7.b. Si le chemin est fonctionnel alors le lien qu'il couvre est aussi fonctionnel sinon le lien est défectueux. Quand il n'y a aucun chemin possible couvrant un lien donné, le lien demeure dans un état inconnu et il est dit potentiellement défectueux. Pour la majorité des liens potentiellement défectueux montrés à la figure 4-7.b, il a été possible de les lier à TDI-TDO et tous les chemins résultants ont été trouvés fonctionnels. Ce n'est pas le cas pour quelques liens

qui n'ont pas pu être liés à TDI et TDO par défaut d'éléments fonctionnels localisés. La figure 4-7.c montre les liens qui restent dans l'état potentiellement défectueux. Cette figure représente en quelque sorte la zone potentiellement défectueuse du réticule ou en d'autres termes la zone susceptible de contenir la ou les défectuosité(s) du réticule. Cette zone a été identifiée suite à l'appel de l'algorithme heuristique HEURISTIC_1.

Interprétation des résultats expérimentaux

L'application de l'algorithme de diagnostic sur le réticule du mini-WaferIC, a permis de montrer qu'il contient plusieurs défectuosités vu qu'il y a six rectangles non fonctionnels (Figure 4-7.a). Les rectangles contiennent 83, 81, 79, 91, 93 et 95 liens intercellulaires soit au total 522 liens. Or parmi ces liens figurent des liens qui ont été caractérisés comme fonctionnels parce qu'ils appartiennent à d'autres chemins rectangulaires fonctionnels. Le nombre de liens fonctionnels appartenant à ces six rectangles non fonctionnels est égal à 156 liens. Par conséquent, les défectuosités figurent parmi 366 (soit 522-156) liens. Chacun de ces liens est susceptible d'être défectueux (potentiellement défectueux). L'application de l'algorithme de dichotomie a permis de rétrécir le nombre de liens potentiellement défectueux (Figure 4-7.b). En effet, leur nombre est passé de 366 (Figure 4-7.a) à 66 (Figure 4-7.b). En faisant appel à l'algorithme heuristique HEURISTIC_1, le nombre de liens potentiellement défectueux est finalement réduit à 24 liens. Après la configuration des rectangles, les défectuosités figuraient parmi 366 liens alors qu'à la fin de l'algorithme de diagnostic les défectuosités figurent parmi 24 liens.

Les tableaux 4.3 et 4.4 ainsi que la figure 4-8 montrent l'évolution du nombre des liens fonctionnels et des liens potentiellement défectueux au cours des étapes du processus de diagnostic qui sont : la configuration des rectangles, l'application de l'algorithme de dichotomie sur chaque rectangle trouvé non fonctionnel et l'application de l'algorithme HEURISTIC_1 sur les segments sauvegardés dans la liste *heuristic_list* et marquants la limite de l'algorithme dichotomique.

L'estimation du nombre de défectuosités dans un réticule (paragraphe 4.1) a donné qu'un réticule contiendrait au plus une seule défectuosité alors que le réticule du mini-WaferIC ainsi diagnostiqué contient plusieurs éléments susceptibles d'être défectueux. Ceci s'explique par le fait que l'estimation établie dans le paragraphe 4.1 est valable pour un réticule qui vient de sortir

de fabrication et que ce réticule diagnostiqué a été longuement manipulé pour plusieurs expériences dans le laboratoire.

L'algorithme de recherche de(s) élément(s) défectueux retourne une zone potentiellement défectueuse. D'après la zone retournée (Figure 4-7.c) on déduit que les cellules (9,15), (10,15), (9,16) et (10,16) sont probablement défectueuses vu que les quatre liens entrants ainsi que les quatre liens sortants de ces cellules sont potentiellement défectueux.

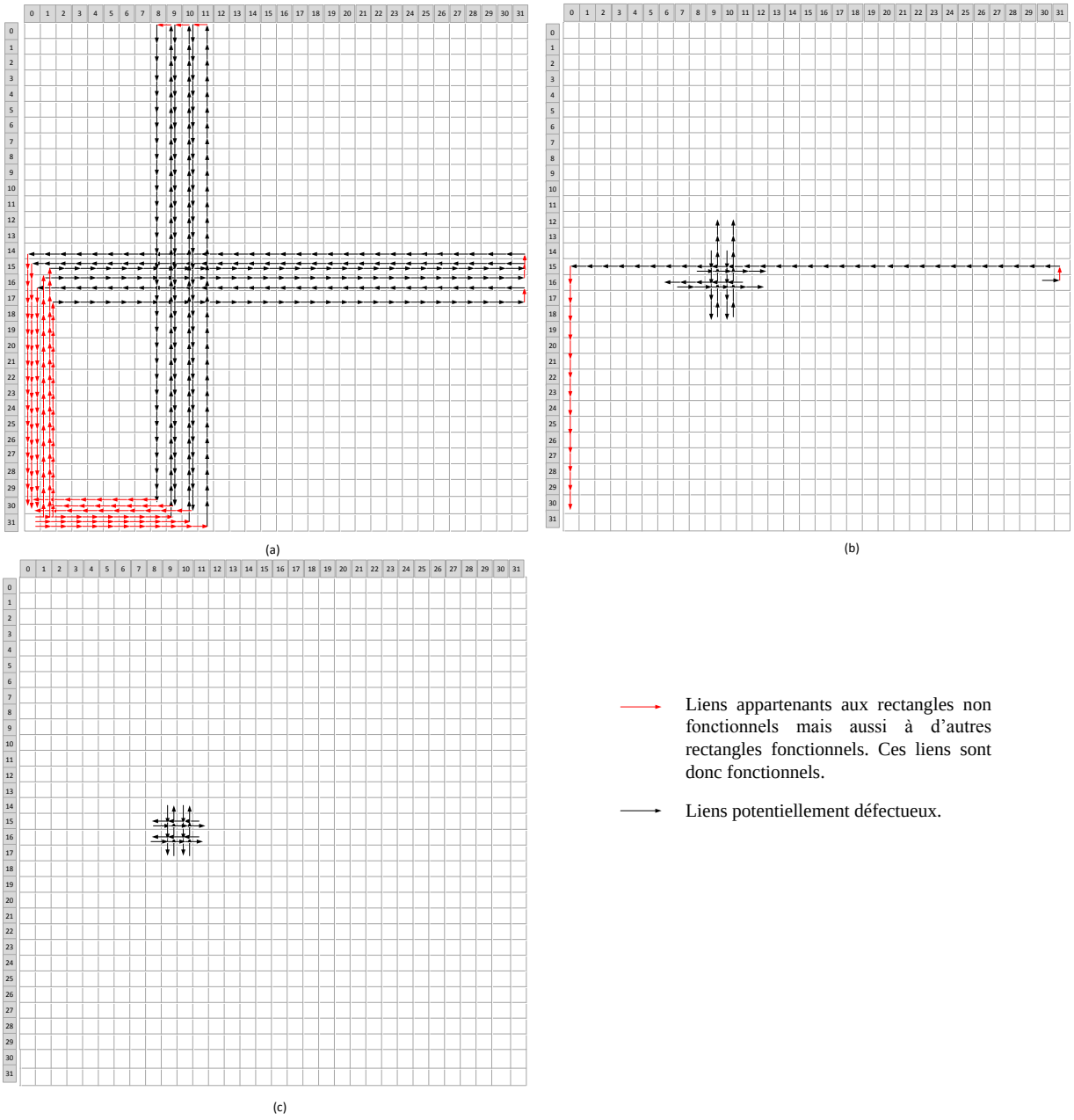


Figure 4-7: Résultat de l'application de l'algorithme de diagnostic sur un réticule du mini-WaferIC (a) Chemins rectangulaires trouvés non fonctionnels (b) Liens potentiellement défectueux trouvés par l'algorithme de dichotomie (c) Liens potentiellement défectueux trouvés après l'appel à l'algorithme HEURISTIC_1.

Ceci peut-être dû à des contrôleurs TAP défectueux de ces cellules par exemple. Une égratignure ou un grain de poussière qui aurait touché ces cellules groupées pourraient être à l'origine du résultat obtenu.

Tableau 4.3 Évolution du nombre de liens potentiellement défectueux durant le diagnostic du réticule du mini-WaferIC.

Nombre de liens potentiellement défectueux			
Avant le diagnostic	Après la création des rectangles	Après l'algorithme de dichotomie	Après les algorithmes dichotomie+HEURISTIC_1
3843 (soit 100%)	366 (soit 9.52%)	66 (soit 1.71%)	24 (soit 0.62%)

Tableau 4.4 Évolution du nombre de liens fonctionnels durant le diagnostic du réticule du mini-WaferIC.

Nombre de liens fonctionnels			
Avant le diagnostic	Après la création des rectangles	Après l'algorithme de dichotomie	Après les algorithmes dichotomie+HEURISTIC_1
0 (soit 0%)	3477 (soit 90.47%)	3777 (soit 98.28%)	3819 (soit 99.3%)

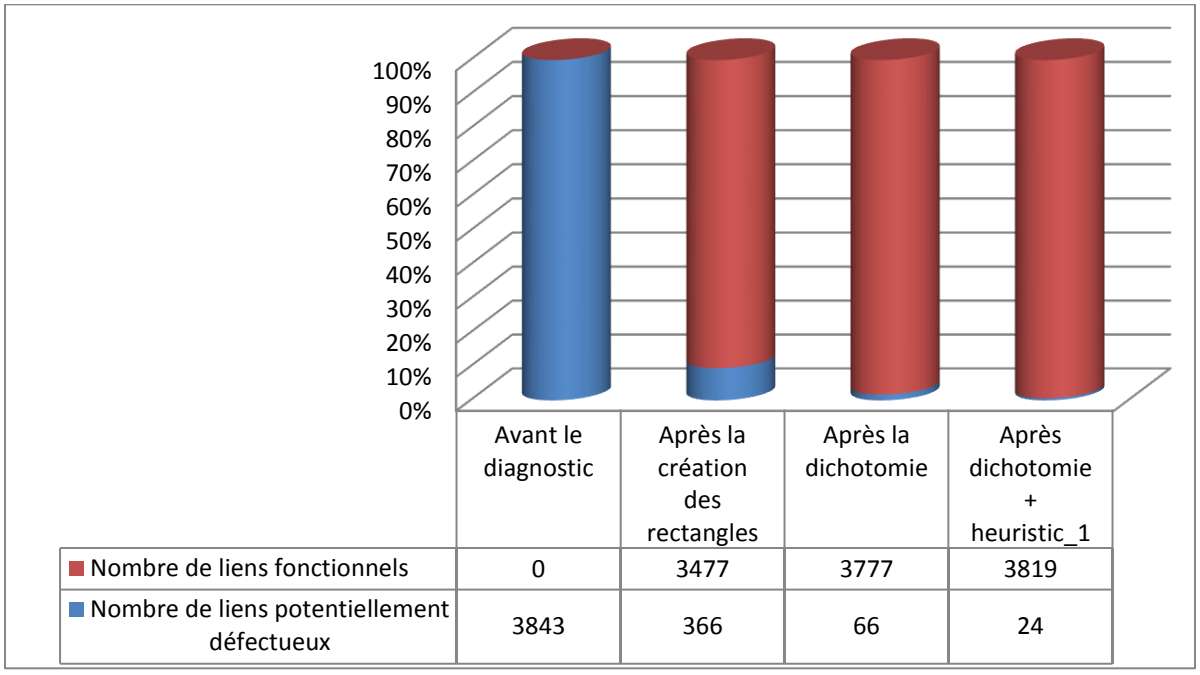


Figure 4-8: Évolution du nombre des liens fonctionnels et des liens potentiellement défectueux durant le diagnostic du réticule du mini-WaferIC.

4.5 Sommaire

Dans ce chapitre les résultats de l'algorithme de diagnostic ont été montrés. Tout d'abord une estimation du nombre de défectuosités que peut contenir un réticule du WaferIC a été faite en se basant sur la loi de Poisson et la loi binomiale négative. Il a été démontré qu'un réticule du WaferIC contiendrait au plus une seule défectuosité à sa sortie de fabrication. Pour tester par simulation l'algorithme de dichotomie, tout d'abord un seul lien défectueux a été injecté. Dans 99.08% des cas l'algorithme retourne le lien injecté après l'appel à l'algorithme de dichotomie. Dans seulement 0.91% des cas il retourne le lien injecté après l'appel des algorithmes HEURISTIC_2 puis HEURISTIC_1. Ceci dépend de l'emplacement du lien défectueux. Les liens nécessitant les algorithmes heuristiques pour être localisés ont été identifiés dans ce chapitre. Pour montrer l'efficacité des algorithmes de diagnostic, jusqu'à 8 liens défectueux ont été injectés aléatoirement dans le réticule. Sur 100 tests effectués, 71 tests localisent les 8 liens avec l'algorithme de dichotomie. Dans les 29 tests restants, quelques liens sont localisés par la dichotomie et quelques autres liens nécessitent l'algorithme HEURISTIC_1 pour être soit exactement localisés ou bien pour repérer la zone où ils se trouvent. L'algorithme a également été testé sur un réticule d'un prototype du WaferIC. Il s'est avéré que ce réticule contient plusieurs défectuosités vu qu'il a été manipulé à plusieurs reprises dans le laboratoire. Initialement, l'état des liens du réticule (3843 liens) est inconnu c'est-à-dire que chaque lien peut aussi bien être fonctionnel que défectueux. Initialement, tous les liens sont donc potentiellement défectueux. Après l'application des algorithmes de diagnostic, une zone contenant 24 liens intercellulaires groupés a été identifiée comme potentiellement défectueuse. Le nombre de liens potentiellement défectueux est donc passé de 3843 à 24 et celui des liens fonctionnels de 0 à 3819.

CONCLUSION

Ce mémoire de maîtrise a présenté tout d'abord la plateforme WaferBoard™ en cours de conception dans le cadre du projet de recherche DreamWafer™. C'est une plateforme de prototypage rapide des systèmes numériques. Une fois mise sur le marché, elle permettrait de réduire le temps requis pour une itération dans un cycle de prototypage de systèmes électroniques d'un temps pouvant excéder 1 mois à quelques minutes. Les coûts liés à la conception et au test de tels systèmes seront aussi réduits. La pièce maitresse de cette plateforme originale est le WaferIC. C'est un circuit intégré à l'échelle de la tranche pour lequel une demande de brevet a été déposée en 2006 par monsieur Richard Norman [4]. Le WaferIC est un substrat intelligent constitué de plusieurs milliers de cellules ayant chacune 4×4 *NanoPads*. Les *NanoPads* sont des plots de l'ordre de quelques micromètres alimentant en tension les broches des CDSP déposés sur la surface active du WaferIC. Le WaferIC assure l'interconnexion entre les puces déposées sur sa surface et ce suivant une *netlist* fournie par l'utilisateur à travers un logiciel dédié au projet qui est le WaferConnect. Le WaferIC est constitué de 76 images de réticules. Chaque réticule est une grille à deux dimensions constituées de 32×32 cellules (au total 1024 cellules par réticule).

Interconnecter les puces revient à programmer les *NanoPads* qui sont en contact avec leurs broches pour les mettre soit en mode alimentation (VDD, GND) ou bien en mode entrée-sortie pour assurer le transfert des signaux entre les différentes puces. Programmer un *NanoPad* revient à programmer la cellule à laquelle il appartient.

Afin de programmer (ou configurer) les cellules qui sont en contact avec les broches, une chaîne conforme au protocole JTAG (établie par Yan Basille-Bellavance en 2009) est configurée depuis un port JTAG externe (TDI). Cette chaîne parcourt les cellules pour les programmer puis ressort par le port de sortie externe (TDO). La chaîne est reconfigurable et tolérante aux pannes. Elle est reconfigurable parce qu'elle peut être reprogrammée pour parcourir n'importe quelles cellules et tolérante aux pannes parce qu'une cellule dont la circuiterie interne est défectueuse peut toujours être évitée ou configurée à partir de sa cellule voisine non défectueuse. Cette fonctionnalité nommée «Contrôle externe» a été introduite par Richard Norman en 2006. Un autre aspect de la

tolérance aux pannes qui n'est toujours pas implémenté, consiste à rendre la chaîne JTAG capable de passer seulement par les éléments (cellules et liens) dont la fonctionnalité est garantie.

Ce mémoire de maîtrise a présenté un algorithme pour diagnostiquer la chaîne JTAG. Cette contribution se situe dans la première étape du processus d'utilisation de la plateforme WaferBoard^{TM} qui est l'étape de diagnostic. Cette étape se fait au démarrage de la plateforme. Le but du diagnostic est de chercher le plus de cellules et de liens intercellulaires fonctionnels possibles permettant de configurer toutes les cellules du WaferIC. L'algorithme développé s'applique sur un réticule mais il pourra par la suite être appliqué sur chacun des 76 réticules pour diagnostiquer le WaferIC au complet.

L'idée est d'envoyer plusieurs flux de bits JTAG à l'intérieur du réticule créant ainsi un ensemble de chemins. Des données sont envoyées à travers chaque chemin et la sortie au niveau du port TDO est observée : Si la sortie coïncide avec ce qui est attendu alors le chemin est dit fonctionnel et toutes les cellules et tous les liens en faisant partie sont caractérisés par l'algorithme comme étant fonctionnels. Dans le cas contraire, c'est-à-dire si le chemin est trouvé non fonctionnel alors l'algorithme conclut qu'il y a une ou plusieurs défectuosités dans ce chemin. Dans ce cas, l'algorithme de diagnostic fait appel à un algorithme de recherche itératif basé sur la dichotomie qui prend le chemin non fonctionnel et le découpe en deux segments, teste chacun des segments et découpe celui trouvé non fonctionnel et ainsi de suite jusqu'à discerner la défectuosité. Pour tester un segment d'un chemin, il faut le compléter par TDI-TDO, c'est-à-dire qu'il faut relier sa première cellule à la cellule TDI et sa dernière cellule à la cellule TDO pour avoir un chemin complet à travers lequel des données peuvent être envoyées. Pour cela, un algorithme basé sur le routage de LEE a également été développé au sein de ce projet de maîtrise pour trouver un chemin fonctionnel entre deux points donnés. Ceci a amené d'autres difficultés. En effet, dans certains cas, l'algorithme de LEE n'est pas capable de trouver des éléments fonctionnels pour compléter les deux segments, l'algorithme de dichotomie fait appel à ce moment à deux algorithmes heuristiques : Un algorithme utilisé dans le cas où il y a un seul lien défectueux et l'autre dans le cas où plusieurs liens défectueux existent dans le réticule. Les algorithmes heuristiques ont pour but de cibler la recherche et localiser le plus étroitement possible le(s) liens(s) défectueux.

À part les algorithmes développés, ce mémoire a présenté une contribution théorique qui consiste à établir une analyse de complexité en temps de diagnostic. Cette analyse a mené à conclure que pour créer un chemin de N cellules, il faut injecter un flux de bits JTAG de longueur $2N^2+18N$. À la lumière de cette étude théorique, il a été conclu qu'il est préférable de couvrir le réticule par un grand ensemble de chemins relativement courts plutôt que par un petit ensemble de longs chemins. Entre deux couvertures possibles qui sont la couverture par des chemins rectangulaires et la couverture par des chemins en serpentin (comparées dans ce mémoire), la couverture par des chemins rectangulaires a été choisie pour notre algorithme de diagnostic.

L'algorithme de diagnostic a été testé sur un réticule d'un prototype du WaferIC. Sur les 1024 cellules du réticule, seulement une zone de 4 cellules a été trouvée comme potentiellement défectueuse. Le reste du réticule a été caractérisé comme fonctionnel.

Des résultats de simulation ont également été extraits. Ils ont démontré que l'algorithme de dichotomie localise exactement le lien défectueux avec un taux de succès de 99.08% si le réticule ne contient qu'un seul lien défectueux. Dans le 0.91% restant, le lien défectueux est localisé par des procédures heuristiques complémentaires.

Les résultats obtenus ne peuvent se comparer à d'autres travaux à cause de l'unicité du projet DreamWafer^{TM} et l'originalité de l'idée principale du JTAG à tête chercheuse qui fut l'objet d'un brevet [23].

Travaux futurs et améliorations :

- L'algorithme de LEE adopté pour connecter un lien donné à TDI-TDO afin de le tester, utilise uniquement des liens caractérisés comme fonctionnels. Par défaut de liens fonctionnels disponibles, le lien potentiellement défectueux ne pourra pas être connecté à TDI-TDO et il reste donc dans un état inconnu. Cependant, une modification au niveau de l'algorithme de LEE, tolérant l'utilisation des liens dont la fonctionnalité n'est pas garantie, pourrait réduire le nombre de liens potentiellement défectueux.
- Comme c'est montré dans le paragraphe 4.3.1, si une défectuosité se trouve au niveau d'un lien situé sur la frontière du réticule ou dans les coins (Figure 4-3), alors les algorithmes heuristiques sont utilisés pour localiser cette défectuosité. Cependant, ce problème peut être

contourné en étendant le diagnostic sur deux réticules voisins au lieu que ça soit sur un seul réticule.

- Lors de la conception du WaferIC, les cellules TDI et TDO ont été placées dans le coin inférieur gauche du réticule. Ceci a causé que certains liens du réticule sont « non diagnosticables » et on ne saura pas s'ils sont fonctionnels ou bien défectueux. Aucune chaîne unidirectionnelle ne pourra passer par ces liens. Le nombre de liens non diagnosticables dans un réticule s'élève à 123 liens. Ce nombre aurait pu être nettement réduit si les cellules TDI et TDO occupaient le centre du réticule.
- L'algorithme développé tend à diagnostiquer les liens intercellulaires seulement. Il serait intéressant par la suite d'améliorer cet algorithme pour diagnostiquer les liens inter-réticulaires c'est-à-dire les liens JTAG liant deux réticules voisins.
- Diagnostiquer tous les réticules en parallèle pour diminuer le temps de diagnostic est aussi une continuation envisageable de ce travail.
- Comme prochaine étape il faudrait développer un algorithme de routage pour créer une longue chaîne de balayage qui configure les cellules et ce en passant par les éléments caractérisés comme fonctionnels lors de la phase de diagnostic.
- Le contrôle externe n'a pas été utilisé dans l'outil développé. Il serait intéressant par la suite de développer des algorithmes de routage afin de créer des chemins passant par les cellules voisines des cellules caractérisées comme défectueuses. Ces chemins servent à configurer les cellules défectueuses à partir de leurs voisines trouvées fonctionnelles par l'algorithme de diagnostic.

REFERENCES

[1] "IEEE Standard for Test Access Port and Boundary-Scan Architecture - Redline," *IEEE Std 1149.1-2013 (Revision of IEEE Std 1149.1-2001) - Redline*, pp. 1-899, 2013.

[2] TSAI M.K, "From PC Multimedia Chipsets to Digital Consumer SOC: Evolution and Challenges," in *Solid-State Circuits Conference, 2006. ASSCC 2006. IEEE Asian*,vol., no.,pp 11-13,13-15 Nov.2006.

[3] G. Poupon et al., "*System on wafer: a new silicon concept in sip*," *Proceedings of the IEEE*, vol. 97,no.,1,January 2009 pp. 60-69.

[4] R.Norman, "Reprogrammable circuit board with alignment-insensitive support for multiple component contact types,"U.S.Patent 8 124 429,December 15,2006.

[5] J. Otterstedt, M. Kuboschek, J. Castagne, et J. Mucha, "A 16.6 cm 2 large area integrated circuit consisting of 9 video signal processors," in *Innovative Systems in Silicon, 1996. Proceedings., Eighth Annual IEEE International Conference on, Austin, Texas,* 1996, pp. 113-123.

[6] M. Rudack et D. Niggemeyer, "Yield enhancement considerations for a single-chip multiprocessor system with embedded DRAM," in *Defect and Fault Tolerance in VLSI Systems, 1999. DFT'99. International Symposium on, Albuquerque,NM*,1999, pp. 31-39.

[7] Y.-M. Lin et al., "Wafer-scale graphene integrated circuit," vol.332, no.6035,pp 1294-1297, 2011.

[8] J. Schemmel, J. Fieres, et K. Meier, "Wafer-scale integration of analog neural networks," in *Neural Networks, 2008. IJCNN 2008.(IEEE World Congress on Computational Intelligence). IEEE International Joint Conference on, Hong Kong*, 2008, pp. 431-438.

[9] W. André, Y. Blaquière, et Y. Savaria, "A Wafer-Scale Rapid Electronic Systems Prototyping Platform", 2011, [en ligne] disponible http://www.intechopen.com/books/advanced-applications-of-rapid-prototyping-technology-in-modern-engineering/a-wafer-scale-rapid-electronic-systems-prototyoing-platform.

[10] M. J. Madou, *Fundamentals of microfabrication: the science of miniaturization*: CRC press, 2002.

[11] D.Cohen, "Stitching design rules for forming interconnect layers," U.S.Patent 6 225 013, Mai 1,2001.

[12] Y. Xu et D. Chung, "Z-Axis anisotropic electrical conductor films in adhesive and standalone forms for electrical interconnection," *Journal of electronic materials*, vol. 28, pp. 1307-1313, 1999.

[13] E. Lepercq, Y. Blaquiere, R. Norman, et Y. Savaria, "Workflow for an electronic configurable prototyping system," in *Circuits and Systems, ISCAS 2009. IEEE International Symposium on*, 2009, pp. 2005-2008.

[14] "IEEE Standard Test Access Port and Boundary Scan Architecture," *IEEE Std 1149.1-2001*, pp. 1-212, 2001.

[15] A. Miczo, *Digital logic testing and simulation*, John Wiley & Sons, 2003.

[16] V. D. Agrawal, K.-T. Cheng, D. D. Johnson, et T. Sheng Lin, "Designing circuits with partial scan," *Design & Test of Computers, IEEE*, vol. 5, pp. 8-15, 1988.

[17] S. Narayanan, R. Gupta, et M. A. Breuer, "Optimal configuring of multiple scan chains," *Computers, IEEE Transactions on*, vol. 42,1993, pp. 1121-1131.

[18] F. F. Hsu, K. M. Butler, et J. H. Patel, "A case study on the implementation of the Illinois scan architecture," in *Test Conference, 2001. Proceedings. International*, 2001, pp. 538-547.

[19] D. H. Baik, K. K. Saluja, et S. Kajihara, "Random Access Scan: A solution to test power, test data volume and test time," in *VLSI Design, International Conference on*, 2004, pp. 883-883.

[20] V. Ramamurthy, "Random Access Scan",Dept.of electrical and Computer engineering,Auburn,AL,2005.

[21] Y. Blaquiere, Y. Basile-Bellavance, S. Berrima, and Y. Savaria, "Design and validation of a novel reconfigurable and defect tolerant JTAG scan chain," in *Circuits and Systems (ISCAS), 2014 IEEE International Symposium on*, 2014, pp. 2559-2562.

[22] Y. Basile-Bellavance, "Conception d'un système de test et de configuration numérique tolérant aux pannes pour la technologie WAFERIC,"M.Sc.A., École Polytechnique de Montréal, QC,Canada,2009.

[23] Y. Blaquière et al., "Methods, apparatus and system to support large-scale micro-systems including embedded and distributed power supply, thermal regulation, multi-

distributedsensors and electrical signal propagation," U.S.Patent 0285 739, October 31,2013.

[24] J. L. Schafer, F. A. Policastri, et R. J. McNulty, "Partner SRLs for improved shift register diagnostics," in *VLSI Test Symposium, 1992.'10th Anniversary. Design, Test and Application: ASICs and Systems-on-a-Chip', Digest of Papers., 1992 IEEE*, 1992, pp. 198-201.

[25] S. Narayanan et A. Das, "An efficient scheme to diagnose scan chains," in *Proc.Int'l Test Conference, 1997*, pp. 704-713.

[26] S. Edirisooriya et G. Edirisooriya, "Diagnosis of scan path failures," *Proc. VLSI Test Symposium (VTS)*, 1995, pp. 0250-0250.

[27] L.M.Huisman, "Segmented scan chains with dynamic reconfigurations," U.S.Patent 7 139 950, November 21,2006.

[28] M. Abramovici, M. A. Breuer, et A. D. Friedman, "Digital systems testing and testable design," *Comp. Sc. Press.–1998.–652 p,* 1990.

[29] S. Jochan, N. Landis, et D. Monson, "Computer-Guided Probing Techniques," in *ITC*, 1981, pp. 253-270.

[30] D. S. Nau, "Expert computer systems," *Computer*, vol. 16, 1983,pp. 63-85.

[31] F. Rubin, "The Lee path connection algorithm," *Computers, IEEE Transactions on*, vol. 100, pp. 907-914, 1974.

[32] N. Laflamme-Mayer, Y. Blaquiere, Y. Savaria, et M. Sawan, "A Configurable Multi-Rail Power and I/O Pad Applied to Wafer-Scale Systems," Circuits and systems I:Regular papers,IEEE transactions on,vol.,61, 2014.

Annexe A - Cas de figure où l'algorithme HEURISTIC_1 est appelé

La grille schématisée dans la figure A-1 contient quatre liens défectueux : (3,3) Nord, (5,3) Est, (5,4) Est et (5,5) Est. Quatre chemins rectangulaires (R1, R2, R3 et R4) passant par ces liens sont par conséquent non fonctionnels. Tout lien appartenant à l'un de ces quatre chemins et n'appartenant à aucun autre chemin fonctionnel est potentiellement défectueux.

Afin de localiser ces liens défectueux, chacun des chemins rectangulaires est d'abord analysé par l'algorithme de dichotomie.

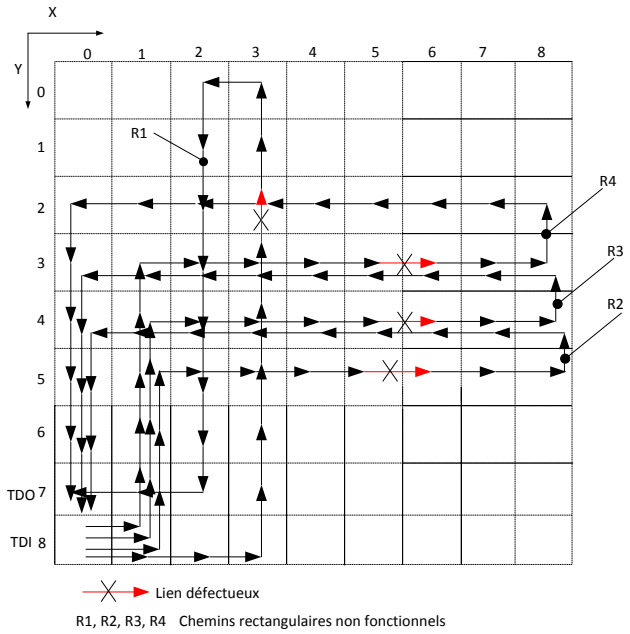


Figure A-1 Exemple de liens défectueux engendrant des chemins rectangulaires non fonctionnels.

1) Analyse du chemin R1 par l'algorithme de dichotomie

Le chemin R1 est divisé en deux segments comme le montre la figure A-2. Le segment 2 est complété par des liens fonctionnels et testé. Le segment 2 est trouvé fonctionnel et par conséquent le segment 1 sera divisé (Figure A-3). Le segment 2 de la figure A-3 est complété par

des liens fonctionnels puis testé. Il a été trouvé non fonctionnel et il sera divisé à son tour (Figure A-4).

On voit bien que ni le segment 1 ni le segment 2 montrés dans la figure A-4 ne peuvent être liés à TDI-TDO. En effet, le segment 1 ne peut pas être lié à TDO car le lien Ouest sortant de la cellule (3,4) est potentiellement défectueux puisqu'il appartient au rectangle non fonctionnel R2 et le lien Est sortant de la cellule (3,4) est potentiellement défectueux car il appartient au rectangle non fonctionnel R3. Le lien Nord sortant de la cellule (3,4) est également potentiellement défectueux car il appartient au segment 1. De même, le segment 2 ne peut pas être lié à TDI car le lien Ouest sortant de la cellule (4,4) est potentiellement défectueux (appartient au rectangle non fonctionnel R2) et le lien Est sortant de la cellule (2,4) est potentiellement défectueux (appartient au rectangle non fonctionnel R3). Le lien Nord sortant de la cellule (3,5) est également potentiellement défectueux car il appartient au segment 1. Ainsi aucun des segments 1 et 2 de la figure A-4 ne peut être testé. Il n'est plus possible de déterminer quel segment est non fonctionnel pour pouvoir le découper. L'analyse du chemin rectangulaire R1 par l'algorithme de dichotomie s'arrête à ce stade. Les liens des segments 1 et 2 à savoir {(3,2) Nord, (3,3) Nord, (3,4) Nord, (3,5) Nord, (3,6) Nord} seront ajoutés à la liste *heuristic_list* afin d'être traités par l'algorithme HEURISTIC_1 après l'analyse des rectangles R2, R3 et R4 par l'algorithme de dichotomie.

2) Analyse des chemins R2, R3 et R4 par l'algorithme de dichotomie

Les chemins rectangulaires R2, R3 et R4 sont par la suite analysés successivement par l'algorithme de dichotomie. Celui-ci identifie les liens (5,5) Est, (5,4) Est et (5,3) Est comme étant les liens défectueux se trouvant respectivement sur les chemins R2, R3 et R4. Tous les autres liens de ces rectangles sont caractérisés comme fonctionnels.

3) Application de l'algorithme HEURISTIC_1 sur les éléments de la liste *heuristic_list*

Une fois l'algorithme de dichotomie est appliqué sur chacun des rectangles non fonctionnels, le nombre de liens fonctionnels se voit élargi. Les segments contenus dans la liste *heuristic_list* seront traités par l'algorithme HEURISTIC_1. Des chemins complets sont créés couvrant respectivement les liens (3,2) Nord, (3,3) Nord, (3,4) Nord, (3,5) Nord, (3,6) Nord. On trouvera

que seul le chemin couvrant le lien (3,3) Nord est trouvé non fonctionnel. Par conséquent, le lien défectueux qui a rendu le rectangle R1 non fonctionnel est bien le lien (3,3) Nord.

Ainsi les quatre liens défectueux de la grille sont localisés.

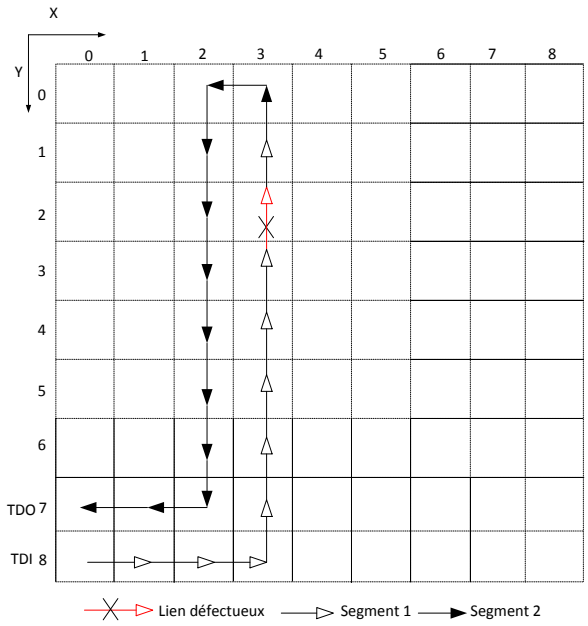


Figure A-2 Division du chemin rectangulaire R1 en deux segments.

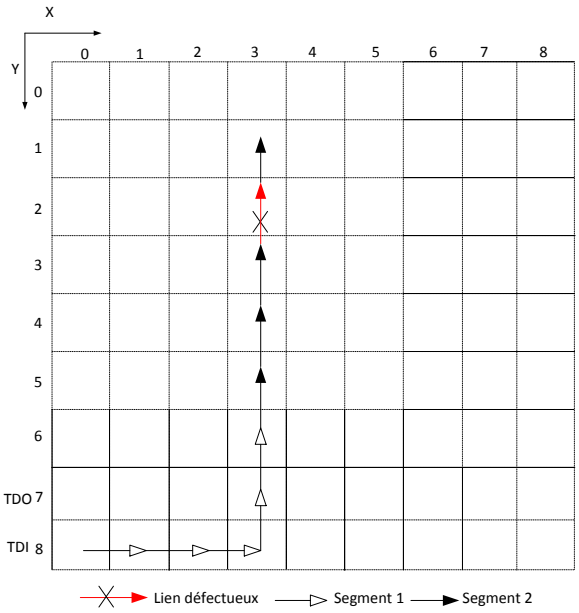


Figure A-3 Division du segment 1 de la figure A-2 en deux segments.

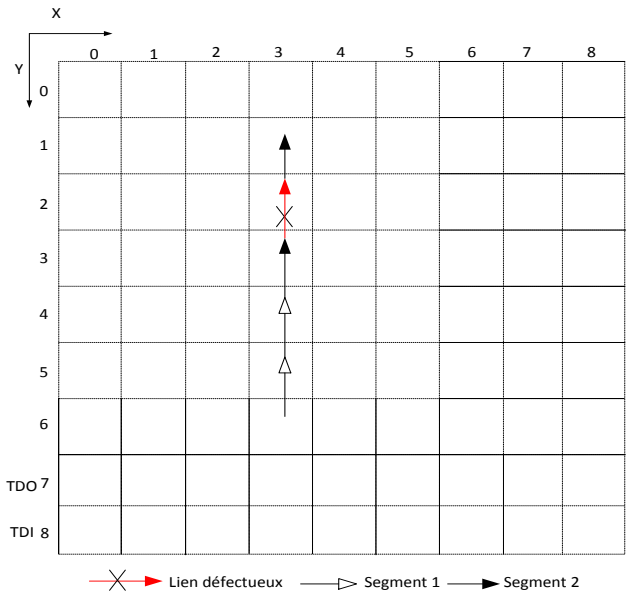


Figure A-4 Division du segment 2 de la figure A-3 en deux segments.

Annexe B - Cas de figure où l'algorithme de dichotomie ne retrouve pas les défectuosités

L'annexe B présente un cas de figure où l'algorithme de dichotomie ne trouve pas les liens défectueux. En effet, afin de localiser une défectuosité se trouvant sur un chemin donné, l'algorithme de dichotomie divise ce chemin en deux segments s_1 et s_2 et teste chacun des segments. Puis divise le segment trouvé non fonctionnel. Et ainsi de suite jusqu'à arriver à la défectuosité. Pour tester un segment donné s_i, l'algorithme de dichotomie doit toujours le lier à TDI et TDO par des liens et des cellules caractérisés comme fonctionnels pour avoir un chemin complet. Il arrive parfois que ni s_1 ni s_2 n'arrivent à se compléter (pas de liens fonctionnels disponibles). Dans ce cas l'algorithme de dichotomie n'est pas capable de localiser le lien défectueux et il fera appel à un algorithme heuristique pour délimiter la zone contenant le lien défectueux. Cette annexe montre un exemple où l'algorithme de dichotomie n'est pas capable de lier les deux segments d'un chemin à TDI et TDO.

Étant donnés deux liens défectueux dans un réticule qui sont :

- Le lien sortant Sud de la cellule (0,2).
- Le lien sortant Nord de la cellule (10,7).

Le lien sortant Sud de la cellule (0,2) étant défectueux, il engendre que 4 rectangles (R1, R2, R3 et R4) deviennent non fonctionnels comme le montrent les figures B.1, B.2, B.3 et B.4.

Le lien sortant Nord de la cellule (10,7) étant défectueux, il engendre qu'un rectangle (R5) devient non fonctionnel comme le montre la figure B.5.

Puisque les rectangles verticaux sont créés et testés en premier alors ils seront aussi analysés en premier par l'algorithme de dichotomie. Et par conséquent les rectangles seront analysés dans cet ordre : R1, R5, R2, R3 et R4.

Analyse par dichotomie du rectangle (R1) pour localiser le lien (0,2) Sud

Le rectangle (R1) est divisé en deux segments s_1 et s_2 comme le montre la figure B.7. L'algorithme de dichotomie tente en premier lieu de compléter s_2 par des cellules et des liens fonctionnels pour avoir un chemin complet. Compléter s_2 revient à lier sa première cellule (1,2) à TDI (0,31) et sa dernière cellule (0,30) à TDO (0,30). Lier la cellule (1,2) à TDI n'est pas possible car :

- Le lien Ouest sortant de la cellule (2,2) est potentiellement défectueux parce qu'il fait partie du rectangle non fonctionnel R2 (qui n'est toujours pas analysé par l'algorithme de dichotomie).
- Le lien Nord sortant de la cellule (1,3) est potentiellement défectueux parce qu'il fait partie du segment s_1.
- Aucun chemin ne peut être adopté pour accéder à la cellule (1,2) sans passer par les liens (2,2) Ouest et (1,3) Nord (tout en respectant le fait qu'une cellule ne doit pas être parcourue deux fois par le même chemin).

Même raisonnement pour le segment s_1. Compléter le segment s_1 se résume à trouver des cellules et des liens fonctionnels pour lier sa première cellule (0,31) à TDI et sa dernière cellule (1,2) à TDO (0,30). Lier la cellule (1,2) à TDO n'est pas possible car :

- Le lien Ouest sortant de la cellule (1,2) est potentiellement défectueux parce qu'il fait partie du rectangle non fonctionnel (R2) (qui n'est toujours pas analysé par l'algorithme de dichotomie).
- Le lien sortant Nord de la cellule (1,2) est potentiellement défectueux parce qu'il fait partie du segment s_2.
- Aucun chemin ne peut être adopté pour lier la cellule (1,2) à TDO sans passer par les liens (1,2) Ouest et (1,2) Nord (tout en respectant le fait qu'une cellule ne doit pas être parcourue deux fois par le même chemin).

Analyse par dichotomie du rectangle (R5) pour localiser le lien (10,7) Sud

Le rectangle (R5) est divisé en deux segments s_1 et s_2 come le montre la figure B.6. L'algorithme de dichotomie tente en premier lieu de compléter s_2 par des cellules et des liens fonctionnels pour avoir un chemin complet. Compléter s_2 revient à lier sa première cellule (10,2) à TDI (0,31) et sa dernière cellule (0,30) à TDO (0,30). Lier la cellule (10,2) à TDI n'est pas possible car :

- Le lien Ouest sortant de la cellule (11,2) est potentiellement défectueux parce qu'il fait partie du rectangle non fonctionnel R2 (qui n'est toujours pas analysé par l'algorithme de dichotomie).
- Le lien Nord sortant de la cellule (10,3) est potentiellement défectueux parce qu'il fait partie du segment s_1.
- Aucun chemin ne peut être adopté pour accéder à la cellule (10,2) sans passer par les liens (11,2) Ouest et (10,3) Nord (tout en respectant le fait qu'une cellule ne doit pas être parcourue deux fois par le même chemin).

Même raisonnement pour le segment s_1. Compléter le segment s_1 se résume à trouver des cellules et des liens fonctionnels pour lier sa première cellule (0,31) à TDI (0,31) et sa dernière cellule (10,2) à TDO (0,30). Lier la cellule (10,2) à TDO n'est pas possible car :

- Le lien Ouest sortant de la cellule (10,2) est potentiellement défectueux parce qu'il fait partie du rectangle non fonctionnel (R2) (qui n'est toujours pas analysé par l'algorithme de dichotomie).
- Le lien sortant Nord de la cellule (10,2) est potentiellement défectueux parce qu'il fait partie du segment s_2.
- Aucun chemin ne peut être adopté pour lier la cellule (10,2) à TDO sans passer par les liens (10,2) Ouest et (10,2) Nord (tout en respectant le fait qu'une cellule ne doit pas être parcourue deux fois par le même chemin).

Analyse par dichotomie du rectangle (R2) pour localiser le lien (0,2) Sud

itération	Chemin à diviser en deux	Segment s1	Segment s2	Résultat du test de s1	Résultat du test de s2
1	R2	De (0,31) à (30,3)	De (30,3) à (0,30)	fonctionnel	Non fonctionnel
2	De (30,3) à (0,30)	De (30,3) à (4,2)	De (4,2) à (0,30)	fonctionnel	Non fonctionnel
3	De (4,2) à (0,30)	De (4,2) à (0,13)	De (0,13) à (0,30)	Non fonctionnel	fonctionnel
4	De (4,2) à (0,13)	De (4,2) à (0,5)	De (0,5) à (0,13)	Non fonctionnel	fonctionnel
5	De (4,2) à (0,5)	De (4,2) à (1,2)	De (1,2) à (0,5)	fonctionnel	Non fonctionnel
6	De (1,2) à (0,5)	De (1,2) à (0,2)	De (0,2) à (0,5)	Impossible à lier	Impossible à lier

Analyse par dichotomie du rectangle (R3) pour localiser le lien (0,2) Sud

itération	Chemin à diviser en deux	Segment s1	Segment s2	Résultat du test de s1	Résultat du test de s2
1	R3	De (0,31) à (30,2)	De (30,2) à (0,30)	fonctionnel	Non fonctionnel
2	De (30,2) à (0,30)	De (30,2) à (3,1)	De (3,1) à (0,30)	fonctionnel	Non fonctionnel
3	De (3,1) à (0,30)	De (3,1) à (0,13)	De (0,13) à (0,30)	Non fonctionnel	fonctionnel
4	De (3,1) à (0,13)	De (3,1) à (0,5)	De (0,5) à (0,13)	Non fonctionnel	fonctionnel
5	De (3,1) à (0,5)	De (3,1) à (0,1)	De (0,1) à (0,5)	Impossible à lier	Impossible à lier

Analyse par dichotomie du rectangle (R4) pour localiser le lien (0,2) Sud

itération	Chemin à diviser en deux	Segment s1	Segment s2	Résultat du test de s1	Résultat du test de s2
1	R4	De (0,31) à (30,1)	De (30,1) à (0,30)	fonctionnel	Non fonctionnel
2	De (30,1) à (0,30)	De (30,1) à (3,0)	De (3,0) à (0,30)	fonctionnel	Non fonctionnel
3	De (3,0) à (0,30)	De (3,0) à (0,12)	De (0,12) à (0,30)	Non fonctionnel	fonctionnel
4	De (3,0) à (0,12)	De (3,0) à (0,4)	De (0,4) à (0,12)	Non fonctionnel	fonctionnel
5	De (3,0) à (0,4)	De (3,0) à (0,0)	De (0,0) à (0,4)	Impossible à lier	Impossible à lier

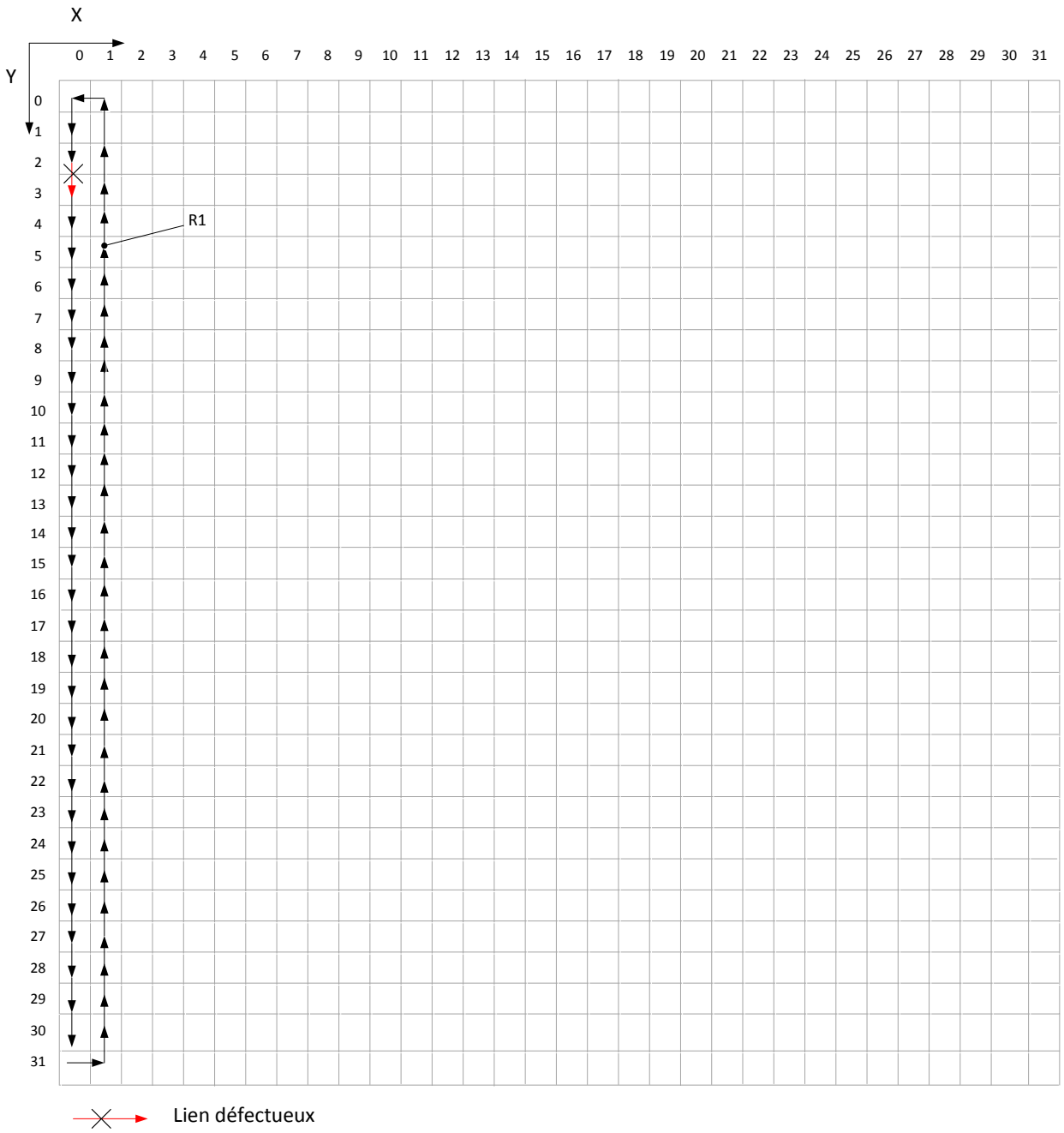


Figure B-1: Un rectangle vertical non fonctionnel (R1) à cause d'un lien défectueux (Lien Sud de la cellule (0,2)).

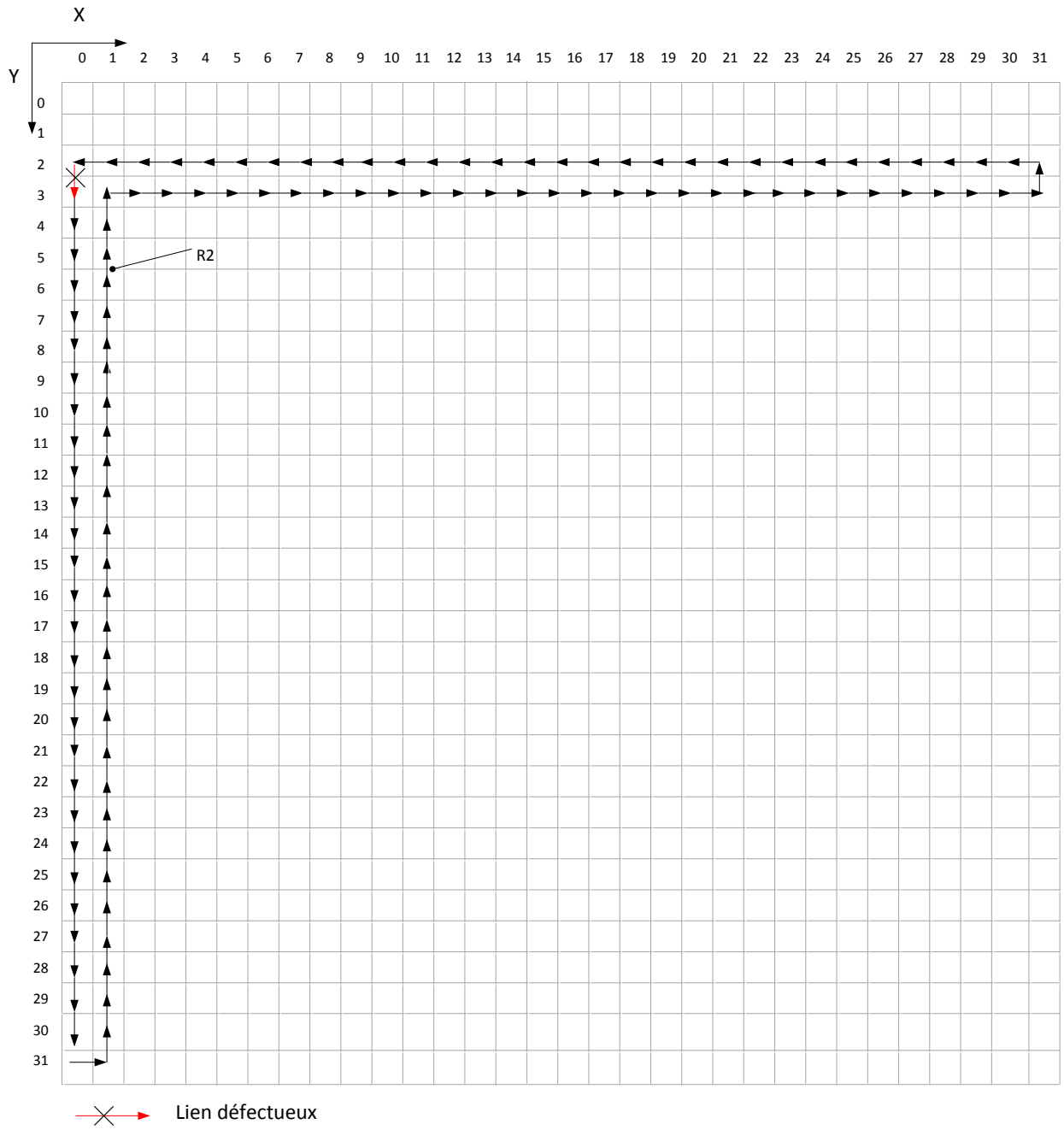


Figure B-2: Un rectangle horizontal non fonctionnel (R2) à cause d'un lien défectueux (Lien Sud de la cellule (0,2)).

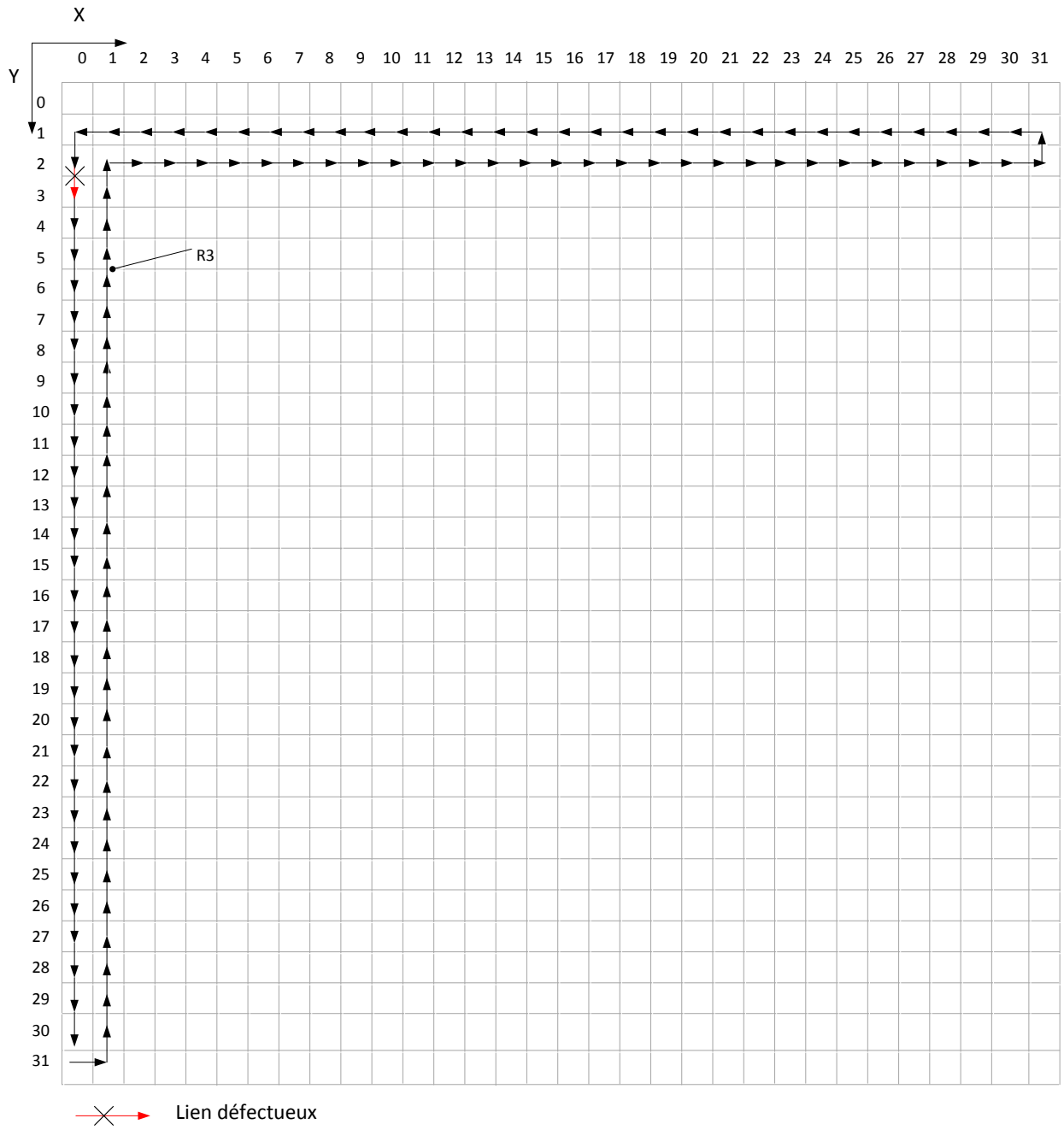


Figure B-3: Un rectangle horizontal non fonctionnel (R3) à cause d'un lien défectueux (Lien Sud de la cellule (0,2)).

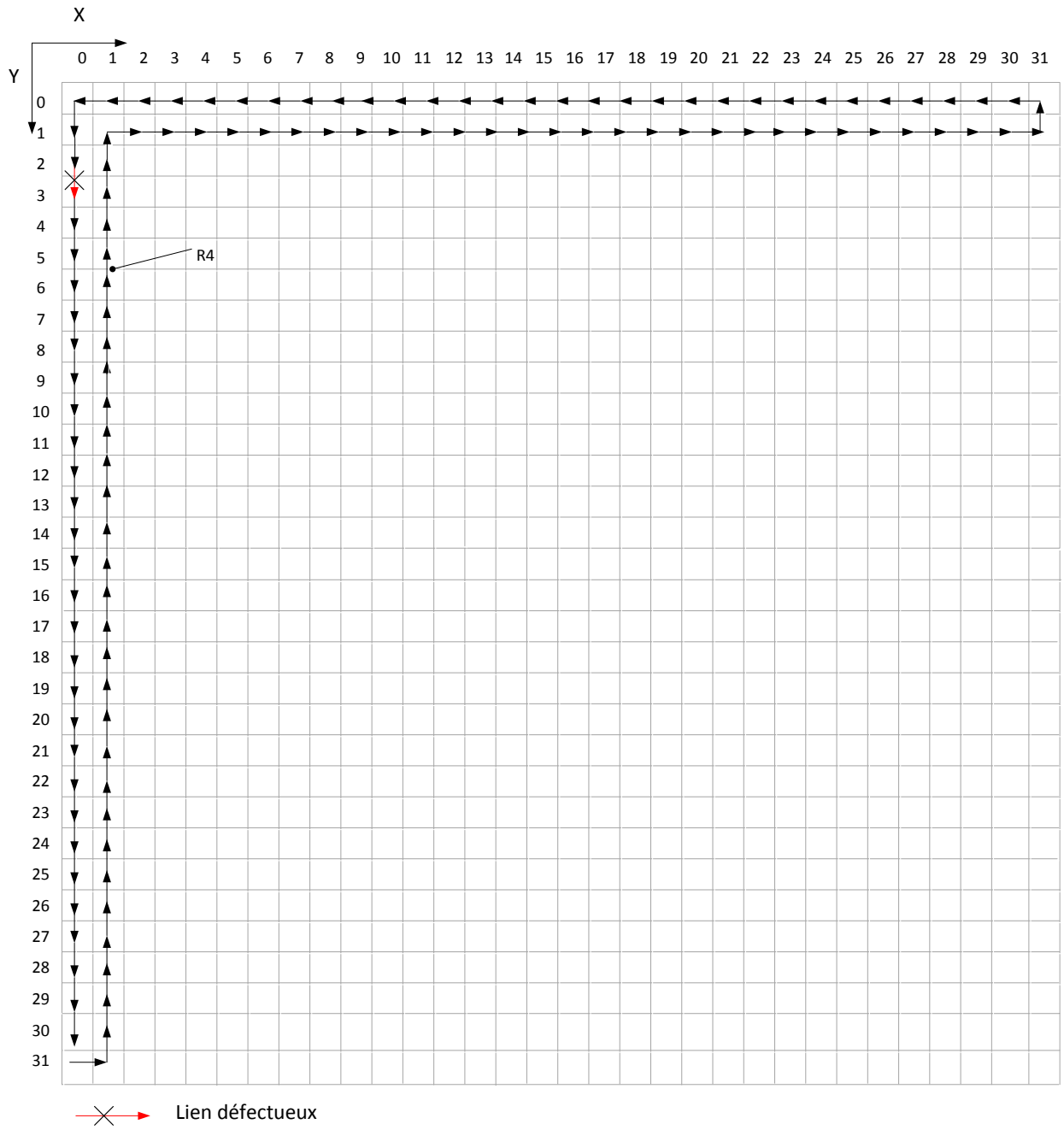


Figure B-4: Un rectangle horizontal non fonctionnel (R4) à cause d'un lien défectueux (Lien Sud de la cellule (0,2)).

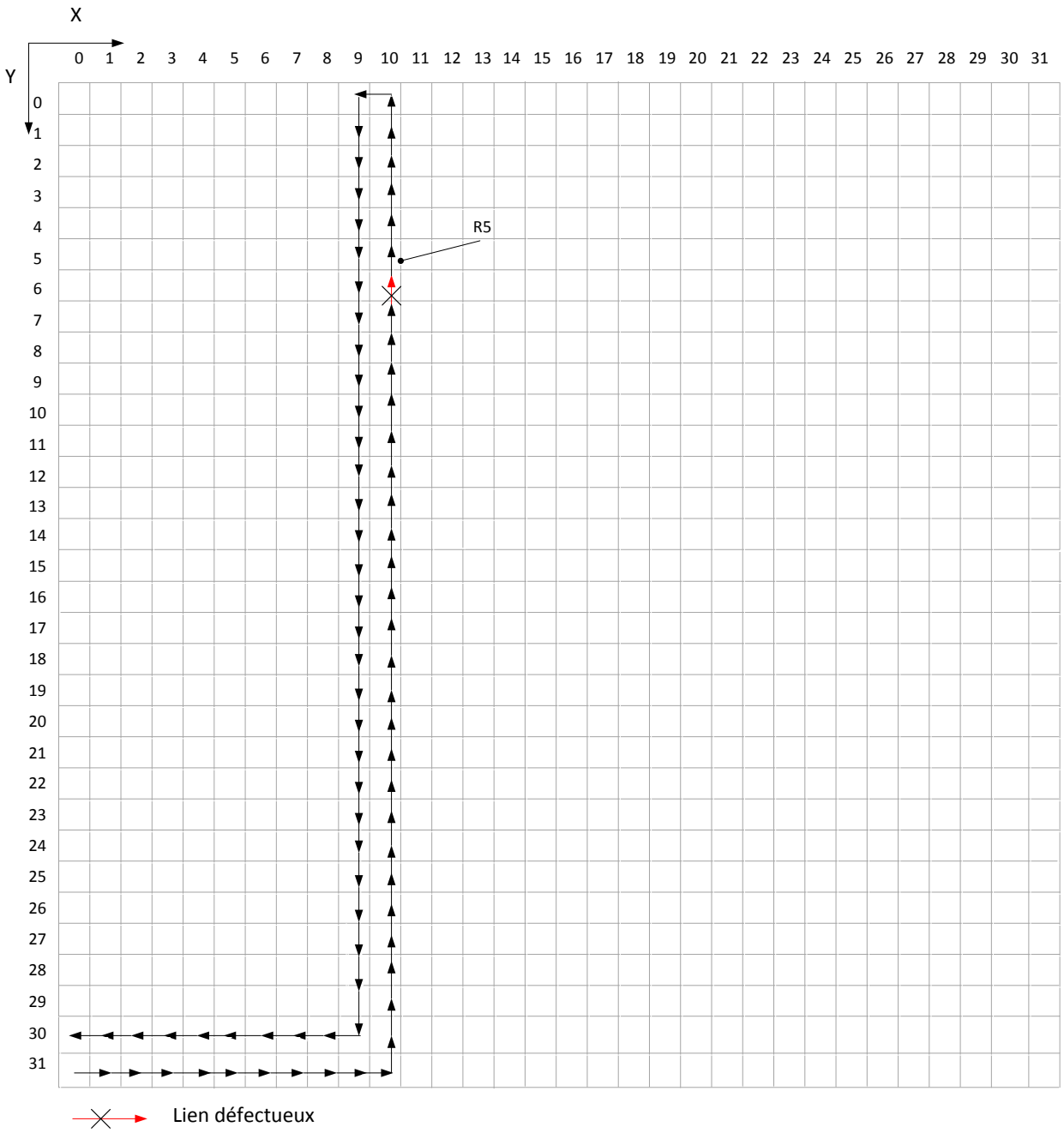


Figure B-5: Un rectangle vertical non fonctionnel (R5) à cause d'un lien défectueux (Lien Nord de la cellule (10,7)).

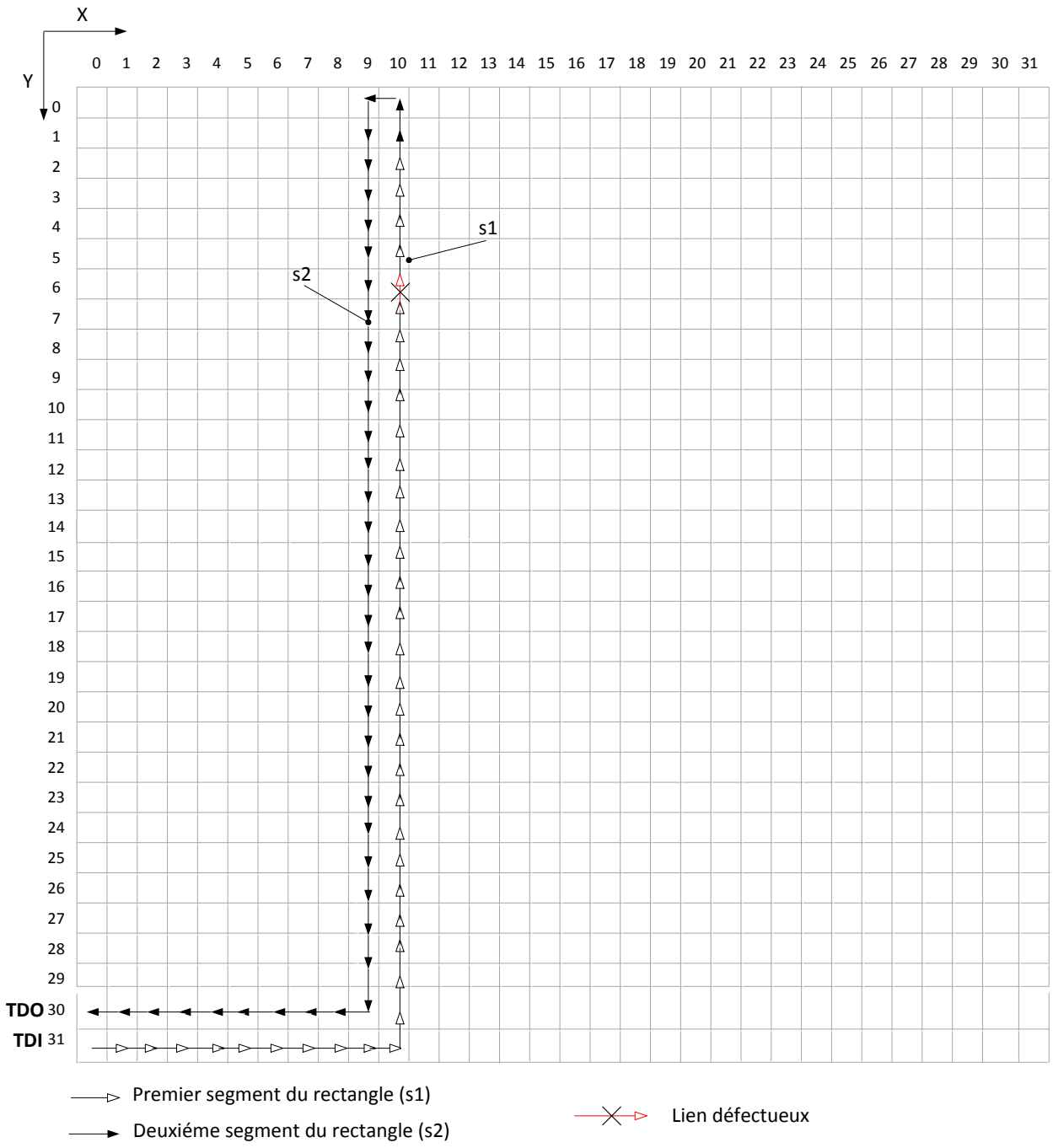


Figure B-6: Division du rectangle (R5) en deux segments s1 et s2.

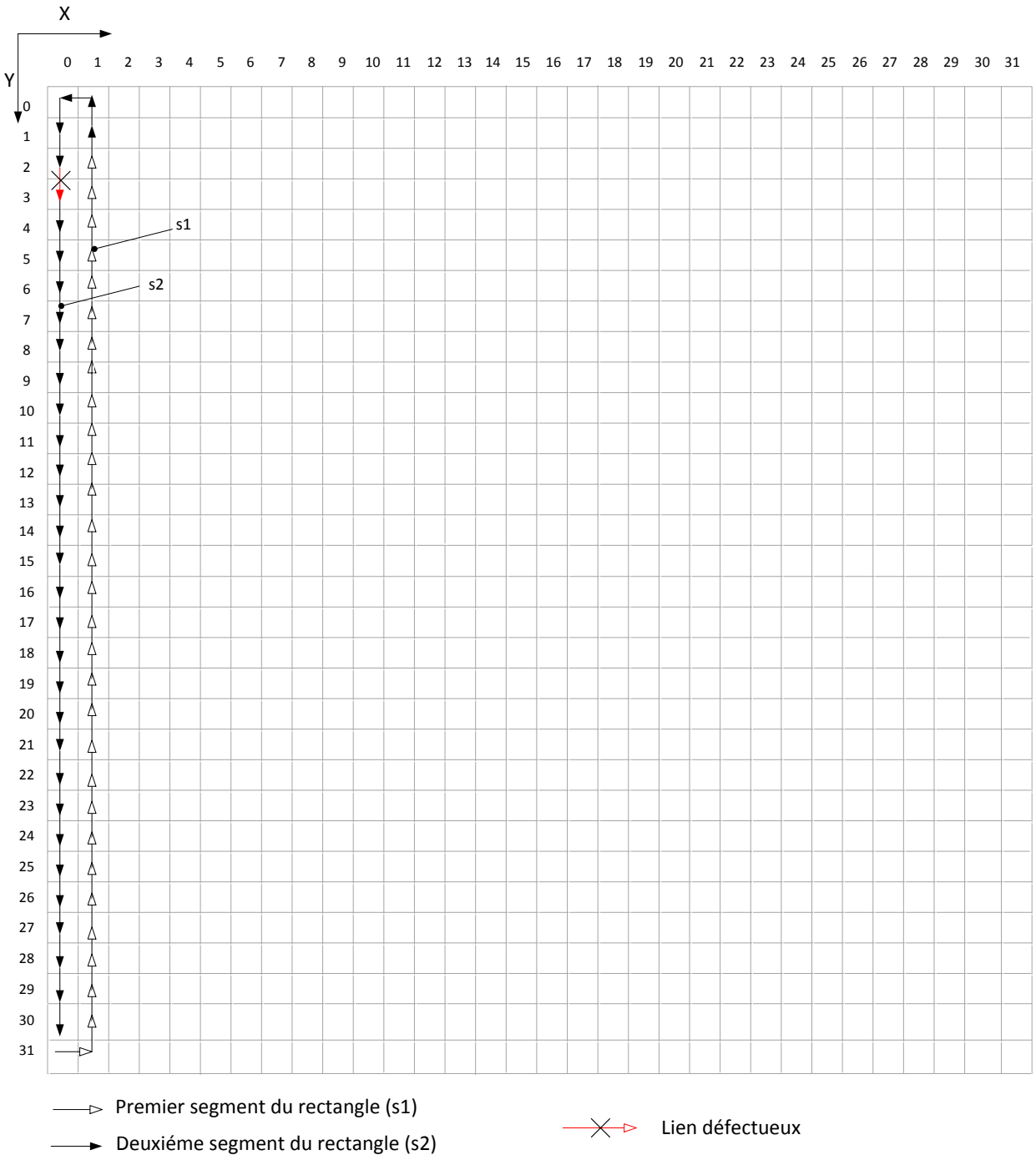


Figure B-7: Division du rectangle (R1) en deux segments s1 et s2.